CHAPTER 3

Simulation Software

第 3 章
仿真软件

Recommended sections for a first reading: 3.1 through 3.4

建议首先阅读的节：3.1 节至 3.4 节

3.1 INTRODUCTION

In studying the simulation examples in Chaps. 1 and 2, the reader probably noticed several features needed in programming most discrete-event simulation models, including:

- Generating random numbers, that is, observations from a U(0,1) probability distribution
- Generating random variates from a specified probability distribution (e.g., exponential)
- Advancing simulated time
- Determining the next event from the event list and passing control to the appropriate block of code
- Adding records to, or deleting records from, a list
- Collecting output statistics and reporting the results
- Detecting error conditions

As a matter of fact, it is the commonality of these and other features to most simulation programs that led to the development of special-purpose simulation packages. Furthermore, we believe that the improvement and greater ease of use of these packages have been major factors in the increased popularity of simulation in recent years.

We discuss in Sec. 3.2 the relative merits of using a simulation package rather than a programming language such as C, C++, or Java for building simulation models. In Sec. 3.3 we present a classification of simulation software, including

在学习第 1 章和第 2 章中的仿真例子时，读者可能注意到了大多数离散事件仿真模型编程中所需要的几个要素，包括：

- 产生随机数，即均匀概率分布 U(0,1) 的观测值
- 产生一个特定概率分布（例如，指数分布）的随机变量
- 推进仿真时间
- 从事件列表中确定下一事件，并将控制权转交给适当的代码块
- 向一个列表添加记录或从列表中删除记录
- 搜集输出统计信息并生成结果报告

• 探测错误发生的条件

事实上,这些特征对大多数仿真程序来说是通用的,这就导致了专用仿真软件包的开发。我们认为,近年来仿真之所以日益流行,主要原因是这些软件包的改进与其更易于使用。

在3.2节我们将讨论使用仿真程序包建立仿真模型相对于使用编程语言(如C、C++或Java)的优势;在3.3节我们给出仿真软件的分类,包括对通用的和面向应用的仿真软件包的讨论;3.4节介绍仿真软件包的期望特性,包括动画;3.5节给出*n*个流行的通用仿真软件包——Arena、Extend、Sim和Simio的简要描述,并分别利用这*n*个仿真软件包构建了一个小工厂的仿真模型;在3.6节,我们介绍面向对象的仿真软件;最后,在3.7节,我们介绍若干面向不同应用的仿真软件包。

刊物*OR/MS Today*往往定期地进行仿真软件的评述。

a discussion of general-purpose and application-oriented simulation packages. Desirable features for simulation packages, including animation, are described in Sec. 3.4. Section 3.5 gives brief descriptions of Arena, ExtendSim, and Simio, which are popular general-purpose simulation packages. A simulation model of a small factory is also given for each package. In Sec. 3.6 we describe object-oriented simulation software. Finally, in Sec. 3.7 we delineate a number of different application-oriented simulation packages.

The publication *OR/MS Today* has a survey of simulation software on a fairly regular basis.

3.2 COMPARISON OF SIMULATION PACKAGES WITH PROGRAMMING LANGUAGES

One of the most important decisions a modeler or analyst must make in performing a simulation study concerns the choice of software. If the selected software is not flexible enough or is too difficult to use, then the simulation project may produce erroneous results or may not even be completed. The following are some advantages of using a simulation package rather than a general-purpose programming language:

- Simulation packages automatically provide most of the features needed to build a simulation model (see Secs. 3.1 and 3.4), resulting in a significant decrease in "programming" time and a reduction in overall project cost.
- They provide a natural framework for simulation modeling. Their basic modeling constructs are more closely akin to simulation than are those in a general-purpose programming language like C.
- Simulation models are generally easier to modify and maintain when written in a simulation package.
- They provide better error detection because many potential types of errors are checked for automatically. Since fewer modeling constructs need to be included in a model, the chance of making an error will probably be smaller. (Conversely, errors in a new version of a simulation package itself may be difficult for a user to find, and the software may be used incorrectly because documentation is sometimes lacking.)

On the other hand, some simulation models (particularly for defense-related applications) are still written in a general-purpose programming language. Some advantages of such a choice are as follows:

- Most modelers already know a programming language, but this is often not the case with a simulation package.
- A simulation model efficiently written in C, C++, or Java may require less execution time than a model developed in a simulation package. This is so because a simulation package is designed to address a wide variety of systems with one set of modeling constructs, whereas a C program can be more closely tailored to a particular application. This consideration has, however, become less important with the availability of inexpensive, high-speed PCs.

- Programming languages may allow greater programming flexibility than certain simulation packages.
- The programming languages C++ and Java are object-oriented (see Sec. 3.6), which is of considerable importance to many analysts and programmers, such as those in the defense industry. On the other hand, most simulation packages are not truly object-oriented.
- Software cost is generally lower, but total project cost may not be.

Although there are advantages to using both types of software, we believe, in general, that a modeler would be prudent to give serious consideration to using a simulation package. If such a decision has indeed been made, we feel that the criteria discussed in Sec. 3.4 will be useful in deciding which particular simulation package to choose.

3.3 CLASSIFICATION OF SIMULATION SOFTWARE

In this section we discuss various aspects of simulation packages.

3.3.1 General-Purpose vs. Application-Oriented Simulation Packages

There are two main types of simulation packages for discrete-event simulation, namely, general-purpose simulation software and application-oriented simulation software. A *general-purpose simulation package* can be used for any application, but might have special features for certain ones (e.g., for manufacturing or process reengineering). On the other hand, an *application-oriented simulation package* is designed to be used for a certain class of applications such as manufacturing, health care, or communications networks. A list of application-oriented simulation packages is given in Sec. 3.7.

3.3.2 Modeling Approaches

In the programs in Chaps. 1 and 2, we used the *event-scheduling approach* to discrete-event simulation modeling. A system is modeled by identifying its characteristic events and then writing a set of event routines that give a detailed description of the state changes taking place at the time of each event. The simulation evolves over time by executing the events in increasing order of their time of occurrence. Here a basic property of an event routine is that no simulated time passes during its execution.

On the other hand, most contemporary simulation packages use the process approach to simulation modeling. A *process* is a time-ordered sequence of interrelated events separated by intervals of time, which describes the entire experience of an "entity" as it flows through a "system." The process corresponding to an entity arriving to and being served at a single server is shown in Fig. 3.1. A system or simulation model may have several different types of processes. Corresponding to each

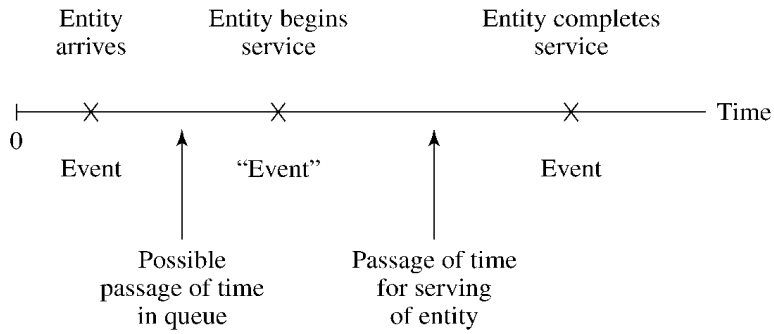

FIGURE 3.1
Process describing the flow of an entity through a system.

process in the model, there is a process "routine" that describes the entire history of its "process entity" as it moves through the corresponding process. A process routine explicitly contains the passage of simulated time and generally has multiple entry points.

To illustrate the nature of the *process approach* more succinctly, Fig. 3.2 gives a flowchart for a prototype customer-process routine in the case of a single-server queueing system. (This process routine describes the entire experience of a customer progressing through the system.) Unlike an event routine, this process routine has multiple entry points at blocks 1, 5, and 9. Entry into this routine at block 1 corresponds to the arrival event for a customer entity that is the most imminent event in the event list. At block 1 an arrival event record is placed in the event list for the *next* customer entity to arrive. (This next customer entity will arrive at a time equal to the time the *current* customer entity arrives plus an interarrival time.) To determine whether the customer entity currently arriving can begin service, a check is made (at block 2) to see whether the server is idle. If the server is busy, this customer entity is placed at the end of the queue (block 3) and is made to wait (at block 4) until selected for service at some undetermined time in the future. (This is called a *conditional wait*.) Control is then returned to the "timing routine" to determine what customer entity's event is the most imminent *now*. (If we think of a flowchart like the one in Fig. 3.2 as existing for each customer entity in the system, control will next be passed to the appropriate entry point for the flowchart corresponding to the most imminent event for some other customer.) When this customer entity (the one made to wait at block 4) is activated at some point in the future (when it is first in queue and another customer completes service and makes the server idle), it is removed from the queue *at block 5* and begins service immediately, thereby making the server busy (block 6). A customer entity arriving to find the server idle also begins service immediately (at block 6); in either case, we are now at block 7. There the departure time for the customer beginning service is determined, and a corresponding event record is placed in the event list. This customer entity is then made to wait (at block 8) until its service has been completed. (This is an *unconditional wait*, since its activation

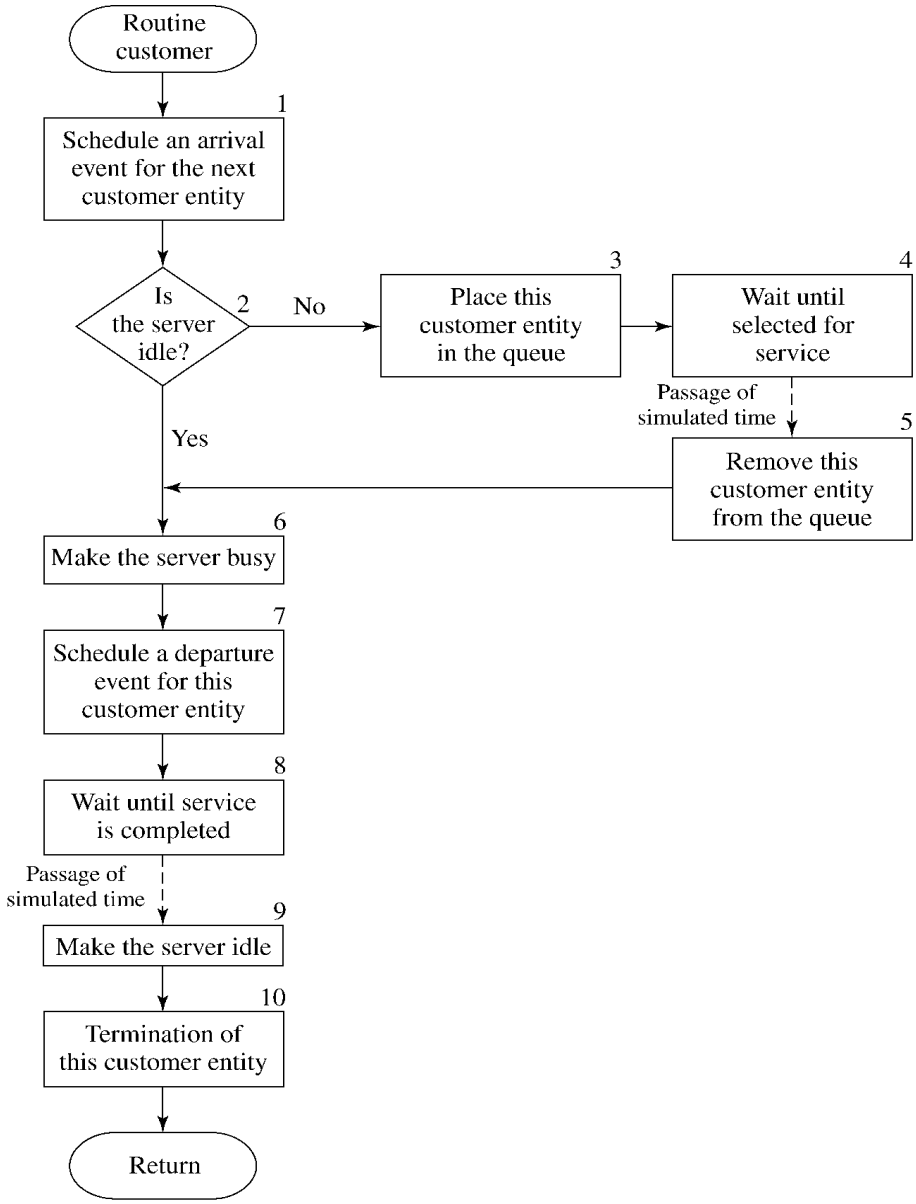

FIGURE 3.2
Prototype customer-process routine for a single-server queueing system.

time is known.) Control is returned to the timing routine to determine what customer entity will be processed next. When the customer made to wait at block 8 is activated at the end of its service, this makes the server idle *at block 9* (allowing the first customer in the queue to become active immediately), and then this customer is removed from the system at block 10.

TABLE 3.1
Entities, attributes, resources, and queues for some common simulation applications

Type of system	Entities	Attributes	Resources	Queues
Manufacturing	Part	Part number, due date	Machines, workers	Queues or buffers
Communications	Message	Destination, message length	Nodes, links	Buffers
Airport	Airplane	Flight number, weight	Runways, gates	Queues
Insurance agency	Application, claim	Name, policy number, amount	Agents, clerks	Queues

A simulation using the process approach also evolves over time by executing the events in order of their time of occurrence. Internally, the process and event-scheduling approaches to simulation are very similar (e.g., both approaches use a simulation clock, an event list, a timing routine, etc.). However, the process approach is more natural in some sense, since one process routine describes the entire experience of the corresponding process entity.

3.3.3 Common Modeling Elements

Simulation packages typically include entities, attributes, resources, and queues as part of their modeling framework. An *entity* (see Table 3.1 for examples) is created, travels through some part of the simulated system, and then is usually destroyed. Entities are distinguished from each other by their *attributes*, which are pieces of information stored with the entity. As an entity moves through the simulated system, it requests the use of *resources*. If a requested resource is not available, then the entity joins a *queue*. The entities in a particular queue may be served in a FIFO (first-in, first-out) manner, served in a LIFO (last-in, first-out) manner, or ranked on some attribute in increasing or decreasing order.

3.4 DESIRABLE SOFTWARE FEATURES

There are numerous features to consider when selecting simulation software. We categorize these features as being in one of the following groups:

- General capabilities (including modeling flexibility and ease of use)
- Hardware and software requirements
- Animation and dynamic graphics
- Statistical capabilities
- Customer support and documentation
- Output reports and graphics

We now discuss each group of features in turn.

3.4.1 General Capabilities

In our opinion, the most important feature for a simulation-software product to have is *modeling flexibility* or, in other words, the ability to model a system whose operating procedures can have any amount of complexity. Note that no two systems are exactly alike. Thus, a simulation package that relies on a *fixed* number of modeling constructs with no capability to do some kind of programming in any manner is bound to be inadequate for certain systems encountered in practice. Ideally, it should be possible to model any system using only the constructs provided in the software—it should not be necessary to use routines written in a programming language such as C. The following are some specific capabilities that make a simulation product flexible:

- Ability to define and change attributes for entities and also global variables, and to use both in decision logic (e.g., if-then-else constructs)
- Ability to use mathematical expressions and mathematical functions (logarithms, exponentiation, etc.)
- Ability to create new modeling constructs and to modify existing ones, and to store them in libraries for use in current and future models

The second most important feature for a simulation product is *ease of use* (and ease of learning), and many contemporary simulation packages have a graphical user interface to facilitate this. The software product should have modeling constructs (e.g., icons or blocks) that are neither too "primitive" nor too "macro." In the former case, a large number of constructs will be required to model even a relatively simple situation; in the latter case, each construct's dialog box will contain an excessive number of options if it is to allow for adequate flexibility. In general, the use of tabs in dialog boxes can help manage a large number of options.

Hierarchical modeling is useful in modeling complex systems. *Hierarchy* allows a user to combine several basic modeling constructs into a new higher-level construct. These new constructs might then be combined into an even higher-level construct, etc. This latter construct can be added to the library of available constructs and can then be reused in this model or future models (see Sec. 3.5.2 for an example). This ability to reuse pieces of model logic increases one's modeling efficiency. Hierarchy is an important concept in a number of simulation packages. It is also a useful way to manage "screen clutter" for a graphically oriented model that consists of many icons or blocks.

The software should have good *debugging aids* such as an interactive debugger. A powerful debugger allows the user to do things such as:

- Follow a single entity through the model to see if it is processed correctly
- See the state of the model every time a particular event occurs (e.g., a machine breakdown)
- Set the value of certain attributes or variables to "force" an entity down a logical path that occurs with small probability

Fast model execution speed is important for certain applications such as large military models and models in which a large number of entities must be processed (e.g., for a high-speed communications network). We programmed a simple manufacturing system in six simulation products and found that, for this model, one product was as much as 11 times faster than another.

It is desirable to be able to develop *user-friendly model "front ends"* when the simulation model is to be used by someone other than the model developer. This capability allows the developer to create an interface by which the nonexpert user can easily enter model parameters such as the mean service time or how long to run the simulation.

Most simulation software vendors offer a *run-time version* of their software, which, roughly speaking, allows the user to change model data but not logic by employing a user-friendly "front end." Applications of a run-time version include:

- Allowing a person in one division of an organization to run a model that was developed by a person in another division who owns a developmental version of the simulation software
- Sales tool for equipment suppliers or system integrators
- Training

Note that a run-time license generally has a considerably lower cost than a normal developmental license or is free.

A feature that is of considerable interest is the ability to *import data from* (and *export data to*) *other applications* (e.g., an Excel spreadsheet or a database).

Traditionally, simulation products have provided performance measures (throughput, mean time in system, etc.) for the system of interest. Now some products also include a *cost module*, which allows costs to be assigned to such things as equipment, labor, raw materials, work in process, finished goods, etc.

In some discrete-event simulations (e.g., steelmaking), it may be necessary to have certain capabilities available from continuous simulation. We call such a simulation a *combined discrete-continuous simulation* (see Sec. 13.4).

Occasionally, one might have a complex set of logic written in a programming language that needs to be integrated into a simulation model. Thus, it is desirable for a simulation package to be able to invoke *external routines.*

It is useful for the simulation package to be easily *initialized in a nonempty and idle state*. For example, in a simulation of a manufacturing system, it might be desirable to initialize the model with all machines busy and all buffers half full, in order to reduce the time required for the model to reach "steady state."

Another useful feature is that *the state of a simulation can be saved at the end of a run* and used to restart easily the simulation at a later time.

Finally, *cost* is usually an important consideration in the purchase of simulation software. Currently, the cost of simulation software for a PC ranges from $1000 to $100,000 or even more. However, there are other costs that must be considered, such as maintenance fees, upgrade fees, and the cost for any additional hardware and software that might be required (see Sec. 3.4.2).

3.4.2 Hardware and Software Requirements

In selecting simulation software, one must consider what *computer platforms* the software is available for. Almost all software is available for Windows-based PCs, and some products are also available for Apple computers. If a software package is available for several platforms, then it should be *compatible across platforms*. The amount of *RAM required* to run the software should be considered as well as *what operating systems are supported*. It is highly desirable if independent replications of a simulation model can be made simultaneously on multiple processor cores or on networked computers.

3.4.3 Animation and Dynamic Graphics

The availability of built-in animation is one of the reasons for the increased use of simulation modeling. In an *animation*, key elements of the system are represented on the screen by icons that dynamically change position, color, and shape as the simulation model evolves through time. (See the Color Plates at the back of the book.) For example, in a manufacturing system, an icon representing a forklift truck will change position when there is a corresponding change in the model, and an icon representing a machine might change color when the machine changes state (e.g., idle to busy) in the model.

The following are some of the uses of animation:

- Communicating the essence of a simulation model (or simulation itself) to a manager or to other people who may not be aware of (or care about) the technical details of the model
- Debugging the simulation computer program
- Showing that a simulation model is *not* valid
- Suggesting improved operational procedures for a system (some things may not be apparent from looking at just the simulation's numerical results)
- Training operational personnel
- Promoting communication among the project team

There are two fundamental types of animation: concurrent and post-processed (also called *playback*). In *concurrent animation* the animation is being displayed at the same time that the simulation is running. Note, however, that the animation is normally turned off when making production runs, because the animation slows down the execution of the simulation. In *post-processed animation*, state changes in the simulation are saved to a disk file and used to drive the graphics *after* the simulation is over. Some simulation software products have both types of animation.

We now discuss desirable features for animation. First, the simulation software should provide *default animation* as part of the model-building process. Since animation is primarily a communications device, it should be possible to *create high-resolution icons* and to save them for later reuse. The software should

come with a *library of standard icons*, or it should be possible to *import icons* from an external source (e.g., Google Warehouse). The software should provide *smooth movement of icons*; icons should not "flash" or "jump." There should be a control to *speed up or slow down the animation*. It should be possible to *zoom* in or out and to *pan* to see different parts of a system too large to fit on one screen. Some software products have *named animation views*, so that one can construct a menu of views corresponding to different parts of the simulated system. It is desirable if the animation uses *vector-based graphics* (pictures are drawn with lines, arcs, and fills) rather than *pixel-based graphics* (pictures are drawn by turning individual pixels on or off). The former type of graphics allows rotation of an object (e.g., a helicopter rotor) as well as a vehicle to maintain its proper orientation as it goes around a corner.

Some simulation products with concurrent animation allow the user to stop the simulation "on the fly" while observing the animation, make changes to certain model parameters (e.g., the number of machines in a workstation), and then instantly restart the simulation. However, this can be *statistically dangerous* if the state of the system and the statistical counters are not reset.

Many simulation packages provide *three-dimensional animation* (the vantage point from which to view the animation can be rotated around all three axes), which might be important for management presentations and for situations in which vertical clearances are important. In these products it may also be possible to provide the viewer of the animation with a perspective of "riding through the system on the back of an entity."

It should be possible to *import CAD drawings and clip art* into an animation.

It is often desirable to display *dynamic graphics and statistics* on the screen as the simulation executes. Examples of dynamic graphics are clocks, dials, level meters (perhaps representing a queue), and dynamically updated histograms and time plots (see Sec. 3.4.6). An example of the latter would be to update a plot of the number in some queue as the simulation moves through time.

3.4.4 Statistical Capabilities

If a simulation product does not have good statistical-analysis features, then it is impossible to obtain correct results from a simulation study. First, the software must have a good *random-number generator* (see Chap. 7), that is, a mechanism for generating independent observations from a uniform distribution on the interval [0, 1]. Note that not all random-number generators found on computers or in software products have acceptable statistical properties. The generator should have at least 100 different streams (preferably far more) that can be assigned to different sources of randomness (e.g., interarrival times or service times) in a simulation model—this will allow different system designs to be compared in a more statistically efficient manner (see Sec. 11.2). The simulation software should produce the same results on different executions if the default seeds are used for the various streams—the seeds should not depend on the internal clock of the computer. On the other hand, the user should be able to set the seed for each stream, if desired.

In general, each source of randomness in the system of interest should be represented in the simulation model by a *probability distribution* (see Chap. 6), not just the perceived mean value. If it is possible to find a standard *theoretical distribution* that is a good model for a particular source of randomness, then this distribution should be used in the model. At a minimum, the following *continuous* distributions should be available: exponential, gamma, Weibull, lognormal, normal, uniform, beta, and triangular. The last distribution is typically used as a model for a source of randomness when no system data are available. Note also that *very few* input random variables in real simulations have a normal distribution. The following *discrete* distributions should also be available: binomial, geometric, negative binomial, Poisson, and discrete uniform.

If a theoretical distribution cannot be found that is a good representation for a source of randomness, then an *empirical* (or *user-defined*) *distribution* based on the data should be used (see Sec. 6.2.4). In this case, random numbers are used to sample from a distribution function constructed from the observed system data.

There should be (a single) command available for making *independent replications* (or *runs*) of the simulation model. This means:

- Each run uses separate sets of different random numbers.
- Each run uses the same initial conditions.
- Each run resets the statistical counters.

Note that simulation results from different runs are independent and also probabilistic copies of each other. This allows (simple) classical statistical procedures to be applied to the results from different runs (see Chap. 9).

There should be a statistically sound method available for constructing a *confidence interval* for a mean (e.g., the mean time in system for a part in a factory). The method should be easy to understand and should provide good statistical results. In this regard, we feel that the *method of replication* (see Secs. 9.4.1 and 9.5.2) is definitely the superior approach.

If one is trying to determine the long-run or "steady-state" behavior of a system, then it is generally desirable to specify a warmup period for the simulation, that is, a point in simulated time when the statistical counters (but not the state of the system) are reset. Ideally, the simulation software should also be able to *determine a value for the warmup period* based on making pilot runs. There is currently at least one simulation product that uses Welch's graphical approach (see Sec. 9.5.1) to specify a warmup period.

It should be possible to construct a *confidence interval for the difference between the means of two simulated systems* (e.g., the current system and a proposed system) by using the method of replication (see Sec. 10.2).

The simulation software should allow the user to specify *what performance measures to collect output data on*, rather than produce reams of default output data that are of no interest to the user.

At least one simulation product allows the user to perform *statistical experimental designs* (see Chap. 12) with the software, such as full factorial designs or fractional factorial designs. When we perform a simulation study, we would like to know what input factors (decision variables) have the greatest impact on the

performance measures of interest. Experimental designs tell us what simulation experiments (runs) to make so that the effect of each factor can be determined. Some designs also allow us to determine interactions among the factors.

A topic that is of interest to some people planning to buy simulation software is "*optimization*" (see Sec. 12.5). Suppose that there are a number of decision variables (input factors) of interest, each with its own range of acceptable values. (There may also be linear constraints on the decision variables.) In addition, there is an objective function to be maximized (or minimized) that is a function of one or more simulation output random variables (e.g., throughput in a manufacturing system) and of certain decision variables. Then the goal of an "optimizer" is to make runs of the simulation model (each run uses certain settings of the decision variables) in an intelligent manner and to determine eventually a combination of the decision variables that produces an optimal or near-optimal solution. These optimization modules use heuristics such as genetic algorithms, simulated annealing, neural networks, scatter search, and tabu search.

3.4.5 Customer Support and Documentation

The simulation software vendor should provide *public training* on the software on a regular basis, and it should also be possible to have *customized training* presented at the client's site. Good *technical support* is extremely important for questions on how to use the software and in case a bug in the software is discovered. Technical support, which is usually in the form of telephone help, should be such that a response is received in at most one day.

Good documentation is a crucial requirement for using any software product. *It should be possible, in our opinion, to learn a simulation package without taking a formal training course.* Generally, there will be a user's guide or reference manual. There should be *numerous detailed examples* available. Most products now have *context-dependent online help*, which we consider very important. (It is not sufficient merely to have a copy of the documentation available in the software.) Several products have a library of "mini examples" to illustrate the various modeling constructs.

There should be a *detailed description of how each modeling construct works*, particularly if its operating procedures are complex. For example, if a simulation-software product for communications networks offers a module for a particular type of local-area network, then its logic should be carefully delineated and any simplifying assumptions made relative to the standard stated.

It is highly desirable to have a *university-quality textbook* available for the simulation package.

Most simulation products offer a *free demo* and, in some cases, a working version of the software can be downloaded from the vendor's website, which will allow small models to be developed and run.

It is useful if the vendor publishes an *electronic newsletter* and has a yearly *users' conference*. The vendor should have *regular updates of the software* (perhaps, once a year).

3.4.6 Output Reports and Graphics

Standard reports should be provided for the estimated performance measures. It should also be possible to *customize reports*, perhaps for management presentations. Since a simulation product should be flexible enough so that it can compute estimates of user-defined performance measures, it should also be possible to write these estimates into a custom report. For each performance measure (e.g., time in system for a factory), the average observed value, the minimum observed value, and the maximum observed value are usually given. If a standard deviation is also given (based on one simulation run), then the user should be sure that it is based on a statistically acceptable method (such as batch means with appropriate batch sizes, as discussed in Sec. 9.5.3), or else it should be viewed as *highly suspect*. [Variance and standard-deviation estimates require independent data, which are rarely produced by one run of a simulation model (see Sec. 4.4).] It should be possible to obtain reports at intermediate points during a simulation run as well as at the end.

The simulation product should provide a variety of (static) graphics. First, it should be possible to make a *histogram* (see Fig. 14.29) for a set of observed data. For continuous (discrete) data, a histogram is a graphical estimate of the underlying probability density (mass) function that produced the data. Time plots are also very important. In a *time plot* (see, for example, Fig. 14.27) one or more key system variables (e.g., the numbers in certain queues) are plotted over the length of the simulation, providing a long-term indication of the dynamic behavior of the simulated system. (An animation provides a short-term indication of the dynamic behavior of a system.) Some simulation products allow the simulation results to be presented in *bar charts* or *pie charts*. Finally, a *correlation plot* (see Fig. 6.29) is a useful way to measure the dependence in the output data produced from one simulation run.

It should be possible to *export individual model output observations* (e.g., times in system) to other software packages such as spreadsheets, databases, statistics packages, and graphical packages for further analysis and display.

3.5 GENERAL-PURPOSE SIMULATION PACKAGES

In Secs. 3.5.1 through 3.5.3 we give brief descriptions of Arena, ExtendSim, and Simio, respectively, which are (at this writing) popular general-purpose simulation packages. In each case we also show how to build a model of a small factory. Section 3.5.4 lists some additional general-purpose simulation packages.

3.5.1 Arena

Arena [see Rockwell (2013) and Kelton et al. (2010)] is a general-purpose simulation package marketed by Rockwell Automation (Wexford, Pennsylvania) that is commonly used for applications such as manufacturing, supply chains, defense, health care, and contact centers. There are two different versions of Arena, namely, the Standard Edition and the Professional Edition.

Modeling constructs, which are called "modules" in Arena, are functionally arranged into a number of "templates." (A module contains logic, a user interface, and, in some cases, options for animation.) The "Basic Process" template contains modules that are used in virtually every model for modeling arrivals, departures, services, and decision logic of entities. The "Advanced Process" template contains modules that are used to perform more advanced process logic and to access external data files in Excel, Access, and SQL databases. The "Advanced Transfer" template contains modules for modeling various types of conveyors, forklift trucks, automated guided vehicles, and other material-handling equipment. The "Flow Process" template is used for modeling tanks, pipes, valves, and batch-processing operations. Also the lower-level "Blocks" and "Elements" templates are used in modeling some complex real-world systems; these two templates constitute what was previously called the SIMAN simulation language.

A model is constructed in Arena by dragging modules into the model window, connecting them to indicate the flow of entities through the simulated system, and then detailing the modules by using dialog boxes or Arena's built-in spreadsheet. A model can have an unlimited number of levels of hierarchy.

"Visual Designer" is used to create concurrent three-dimensional (3-D) animations and "live-data dashboards," which display dynamic graphics (e.g., histograms, pie charts, and time plots). (Two-dimensional animation is also available.) It also allows one to "watch the logic execute" and to perform sophisticated graphical model debugging. AVI files can be generated directly from Arena for sharing animations with other people, and each Arena license includes one additional runtime-only license (see Sec. 3.4.1).

There are an unlimited number of random-number streams (see Chap. 7) available in Arena. Furthermore, the user has access to 12 standard theoretical probability distributions and also to empirical distributions. Arena has a built-in capability for modeling nonstationary Poisson processes (see Sec. 6.12.2), which is a model for entity arrivals with a time-varying rate.

There is an easy mechanism for making independent replications of a particular simulated system and for obtaining point estimates and confidence intervals for performance measures of interest. It is also possible to construct a confidence interval for the difference between the means of two systems. A number of plots are available, such as histograms, time plots, bar charts, and correlation plots. The "OptQuest for Arena" (see Sec. 12.5.2) optimization module is available as an option.

Activity-based costing is incorporated into Arena, providing value-added and non-value-added cost and time reports. Simulation results are stored in a database and are presented using Crystal Reports, which is embedded in Arena.

Microsoft Visual Basic for Applications (VBA) and a complete ActiveX object model are available in Arena. This capability allows more sophisticated control and logic including the creation of user-friendly "front ends" for entering model parameters, the production of customized reports, etc. This technology is also used for Arena's interfaces with many external applications including the Visio drawing package.

Arena Professional Edition includes the ability to create customized modules and to store them in a new template. Arena also has an option that permits a model to run in real time (or any multiple thereof) and to dynamically interact with other

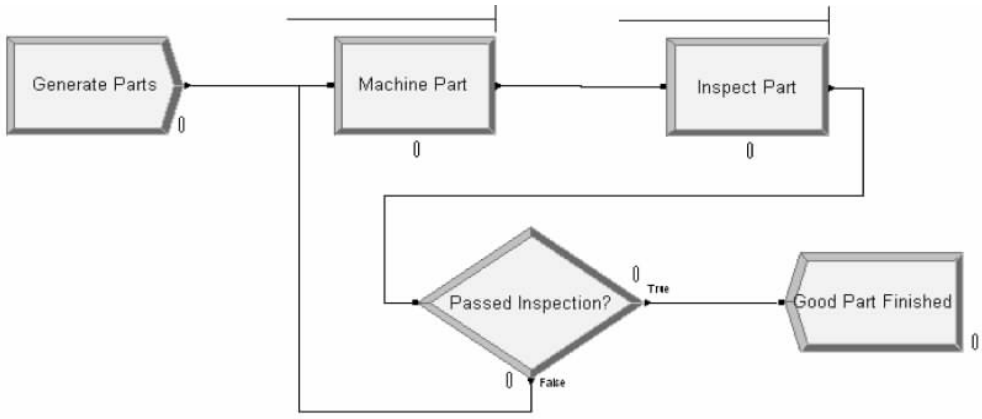

FIGURE 3.3
Arena model for the manufacturing system.

processes; this supports applications such as the High Level Architecture (see Sec. 1.6.2) and testing of hardware/software control systems.

We now develop an Arena model for the simple manufacturing system of Example 9.25, which consists of a machine and an inspector. However, we assume here that the machine never breaks down. Figure 3.3 shows the five required logic modules and the necessary connections to define the entity flow.

The "Create" module, whose dialog box is shown in Fig. 3.4, is used to generate arrivals of parts. We label the module "Generate Parts" and specify that interarrival times are exponentially distributed [denoted "Random (Expo)"] with a mean of 1 minute. The Create module is connected to the "Process" module (see Fig. 3.5), which is

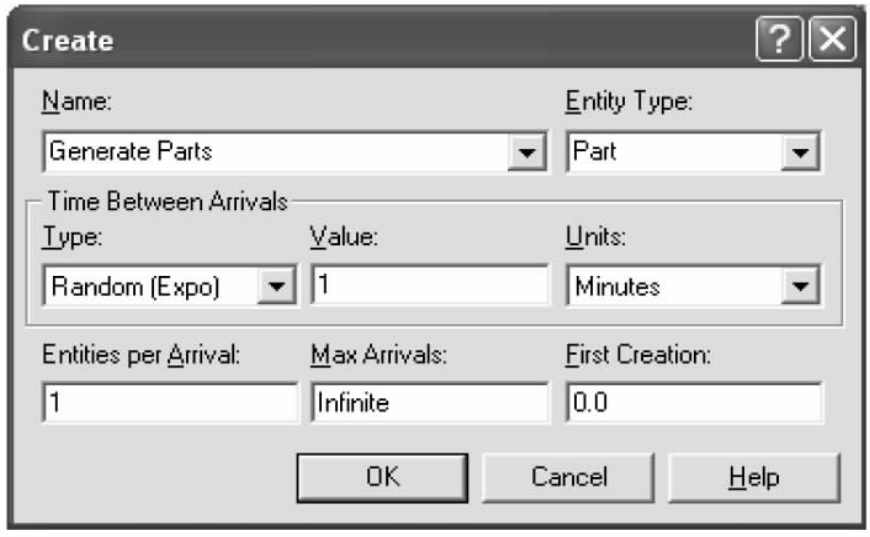

FIGURE 3.4
Dialog box for the Arena Create module "Generate Parts."

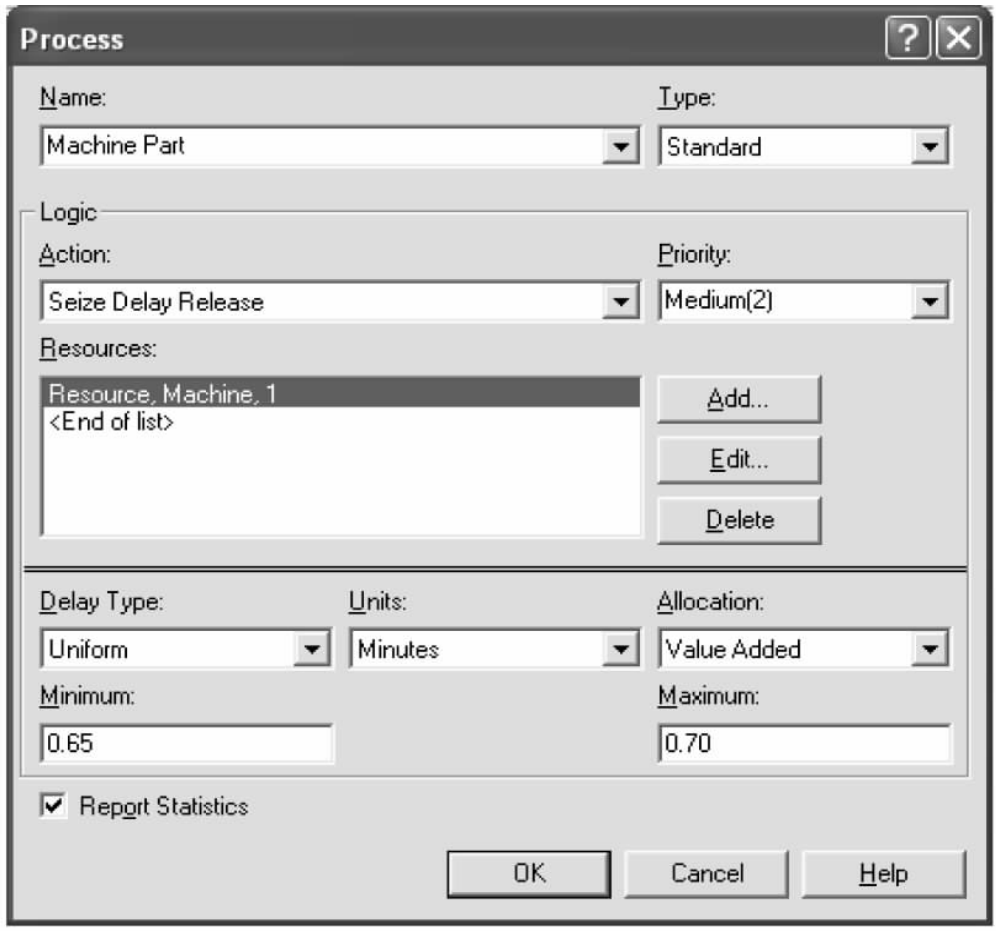

FIGURE 3.5
Dialog box for the Arena Process module "Machine Part."

used to represent the processing of a part at the machine. This module is labeled "Machine Part," has a single resource named "Machine" with one unit, and has processing times that are uniformly distributed between 0.65 and 0.70 minute.

The next Process module (see Fig. 3.6) is used to represent the inspector. We specify that inspection times are uniformly distributed between 0.75 and 0.80 minute. After inspection, a "Decide" module (see Fig. 3.7) specifies that a part can have one of two outcomes: "True" (occurs 90 percent of the time) or "False." If the part is good (True), then it is sent to the "Depart" module (not shown) labeled "Good Part Finished," where it is destroyed. Otherwise (False), it is sent back to the Machine Part module to be remachined.

Finally, we need to use Run > Setup (see Fig. 3.8) to specify the experimental parameters. We state that one run of length 100,000 minutes is desired.

The results from running the simulation are given in Fig. 3.9, from which we see that the average time in system of a part is 4.64 minutes. Additional output statistics can be obtained from the options on the left-hand side of the screen.

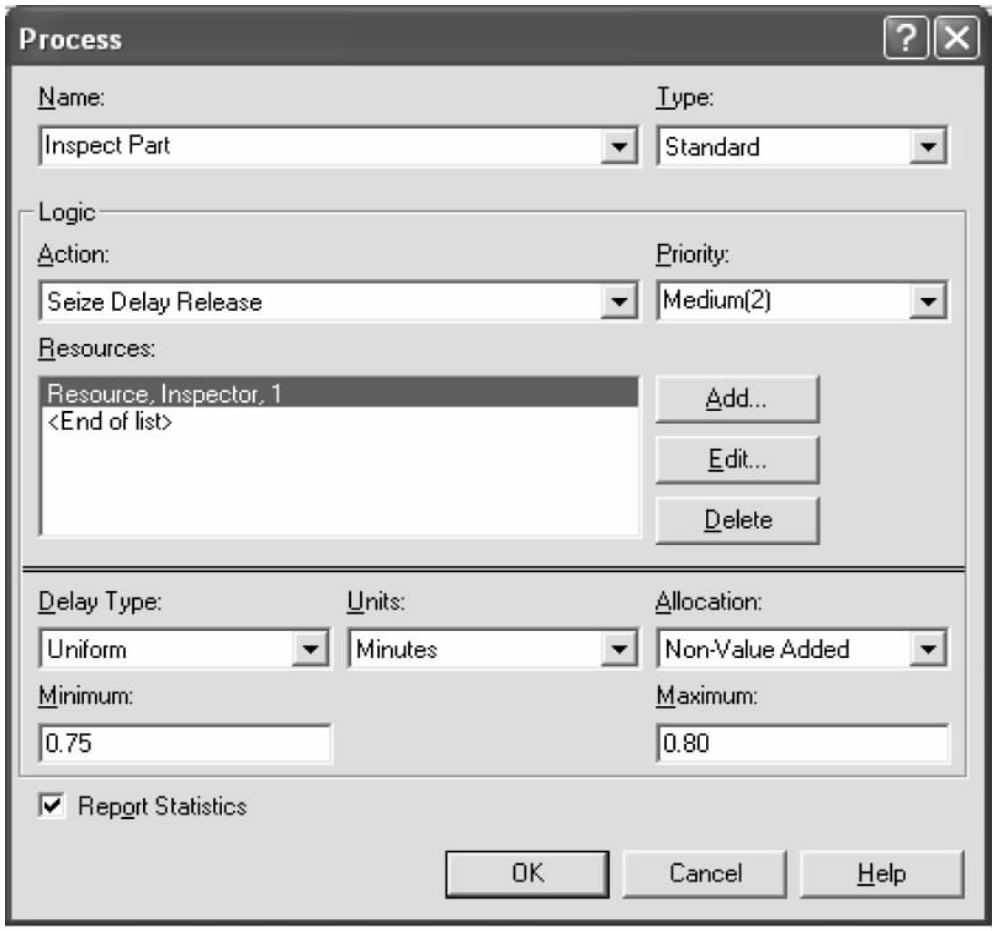

FIGURE 3.6
Dialog box for the Arena Process module "Inspect Part."

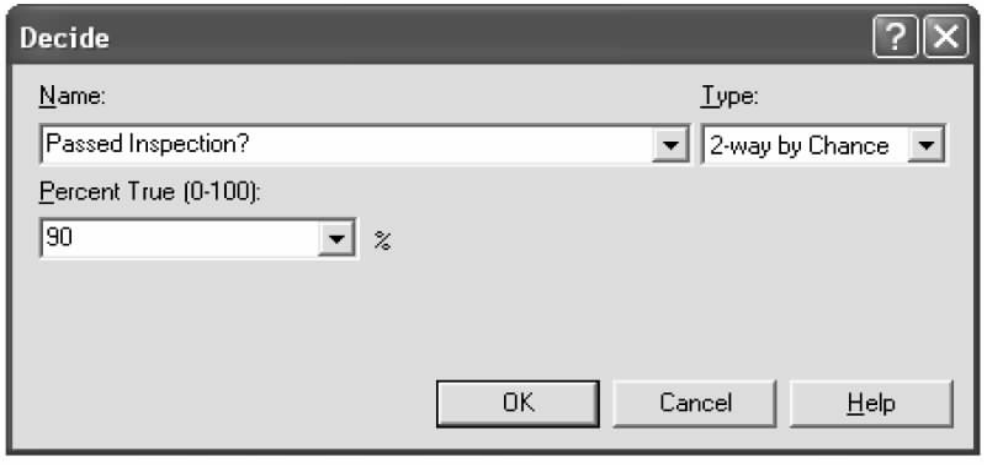

FIGURE 3.7
Dialog box for the Arena Decide module "Passed Inspection?"

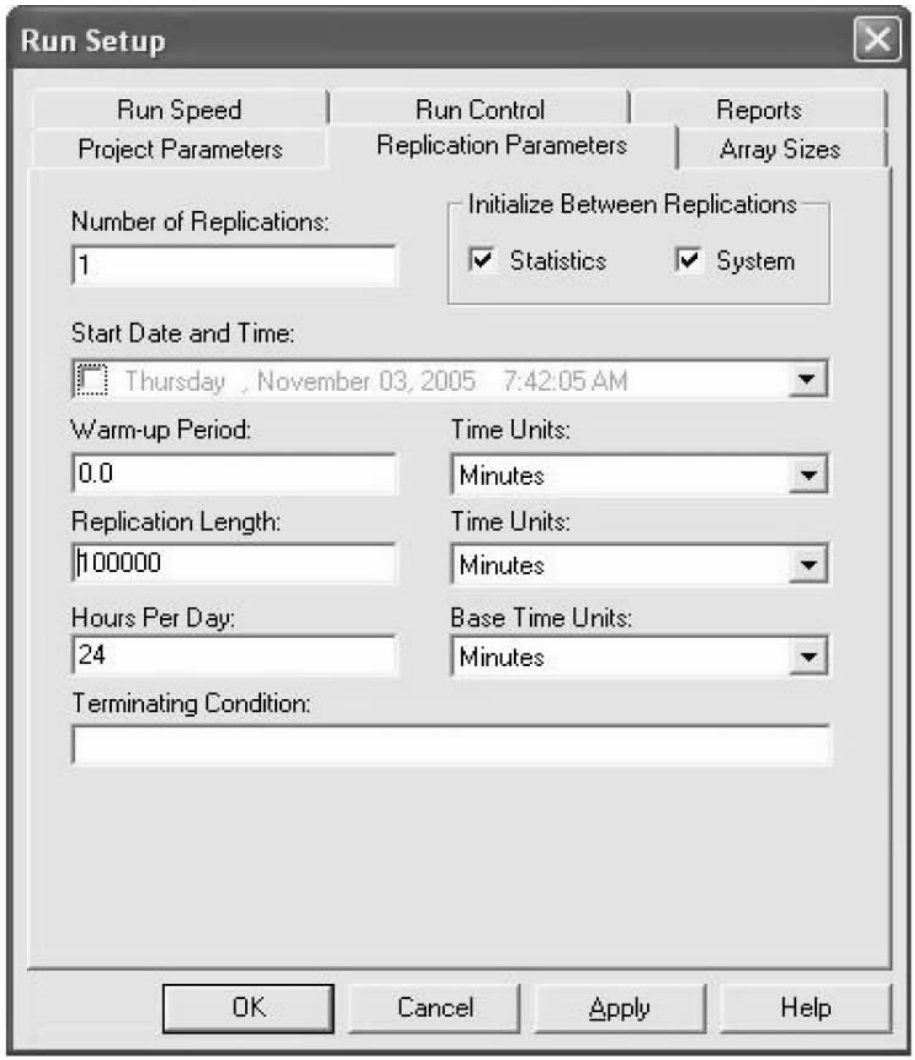

FIGURE 3.8
Dialog box for the Arena Run Setup configuration options.

3.5.2 ExtendSim

ExtendSim [see Imagine (2013)] is the family name for four general-purpose simulation packages marketed by Imagine That, Inc. (San Jose, California). Each ExtendSim product has components aimed at specific market segments, but all products share a core set of features. A model is constructed by selecting blocks from libraries (Item, Value, Plotter, etc.), placing the blocks at appropriate locations in the model window, connecting the blocks to indicate the flow of entities (or values) through the system, and then detailing the blocks using dialog boxes.

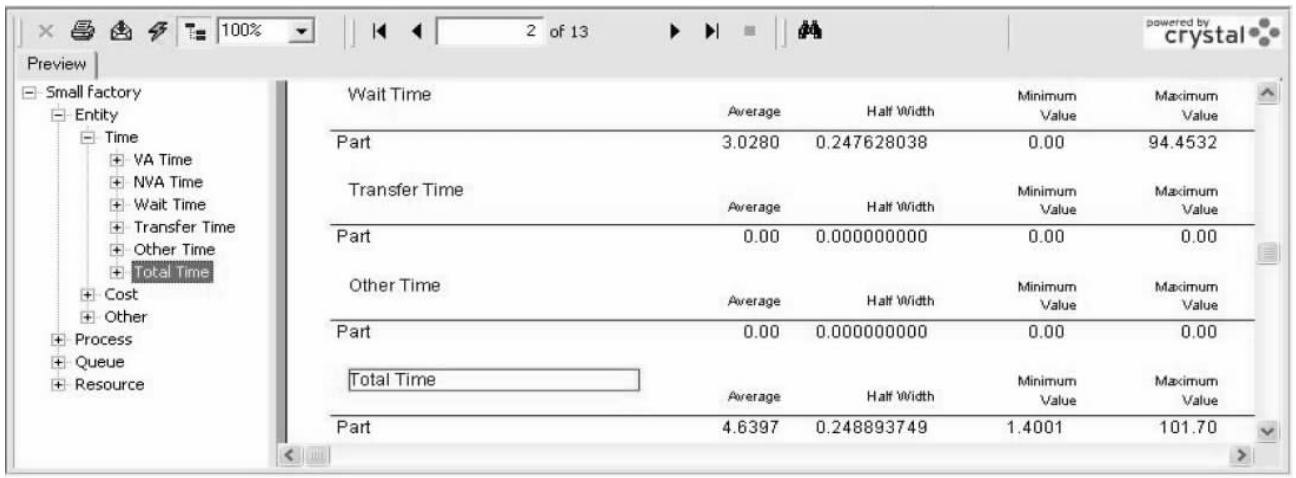

FIGURE 3.9
Simulation results for the Arena model of the manufacturing system.

ExtendSim can model a wide variety of system configurations using the blocks supplied with the product. If needed, the internal ModL language can be used to customize existing blocks and to create entirely new blocks. These "new" blocks can be placed in a new library for reuse in the current model or future models. The code corresponding to a particular block can be viewed by right-clicking on the block and selecting "Open Structure"; this feature is useful for understanding the actual operation of the block. ModL can also access applications and procedures created with external programming languages such as Visual Basic and C++.

A model can have an unlimited number of levels of hierarchy (see below) and also use inheritance (see Sec. 3.6). A "Navigator" allows one to move from one hierarchical level to another. All ExtendSim products provide a basic 2-D animation, and the ExtendSim Suite product also provides 3-D animation.

Each simulation model in ExtendSim has an associated "Notebook," which can contain pictures, text, dialog items, and model results. Thus, a Notebook can be used as a "front end" for a model or as a vehicle for displaying important model results as the simulation is actually running. The parameters for each model can also be stored in, and accessed from, the model's internal relational database; this is useful for data consolidation and management.

There are an essentially unlimited number of random-number streams available in ExtendSim. Furthermore, the user has access to 34 standard theoretical probability distributions and also to empirical distributions. ExtendSim has an easy mechanism for making independent replications of a simulation model and for obtaining point estimates and confidence intervals for performance measures of interest. A number of plots are available such as histograms, time plots, bar charts, and Gantt charts.

There is an activity-based costing capability in ExtendSim that allows one to assign fixed and variable costs to an entity as it moves through a simulated system. For example, in a manufacturing system a part might be assigned a fixed cost for the required raw materials and a variable cost that depends on how long the part spends waiting in queue.

ExtendSim's "Item" library contains blocks for performing discrete-event simulation (entity arrival, service, departure, etc.), as well as for material handling (see Sec. 14.3 for further discussion of material handling) and routing. (An entity is called an "Item" in ExtendSim.) The optional "Rate" library provides blocks for modeling high-speed, high-volume manufacturing systems (e.g., canning lines) within a discrete-event environment. The blocks in the "Value" library are used to perform continuous simulation (see Sec. 13.3) and to provide modeling support (mathematical calculations, simulation-based optimization, data sharing with other applications, etc.) for discrete-event simulation.

ExtendSim's "Scenario Manager" allows the modeler to investigate how the simulation model's responses change from one scenario (a set of values for the model's input parameters or factors) to another. The scenarios of interest can either be entered manually or are specified automatically by the Scenario Manager in the case of a factorial design (see Sec. 12.2). Additionally, the modeler specifies the number of independent replications (each using different random numbers) of each scenario that is desired. The Scenario Manager runs the scenarios iteratively, records the responses for each replication, and the responses are then summarized across the replications for each scenario. The model factors and their corresponding

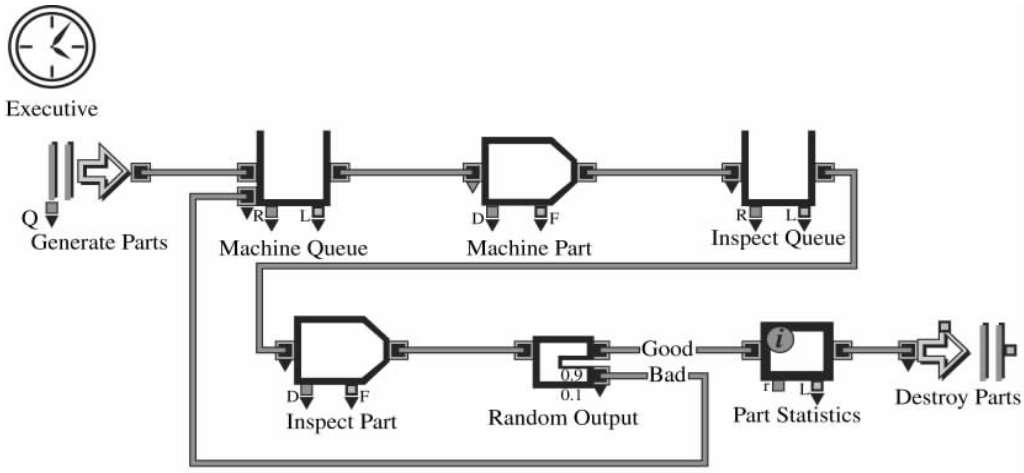

FIGURE 3.10
ExtendSim model for the manufacturing system.

responses can be exported to ExtendSim's database, the JMP and Minitab statistical packages, or Excel for further analysis. ExtendSim also has a built-in optimization module (see Sec. 12.5.2).

We now show how to build an ExtendSim model for the manufacturing system discussed in Sec. 3.5.1. In particular, Fig. 3.10 shows the required blocks and connections for the model; the connections correspond to the flow of entities (parts for this model). All the blocks in this model are from the ExtendSim Item library. We have placed a descriptive label below each block, which we will refer to in the discussion of the model below.

The "Executive" block, which is not graphically connected to any other block, manages the event list for an ExtendSim model. The first block actually in the model is a "Create" block labeled "Generate Parts" (see its dialog box in Fig. 3.11), which is used to generate parts having exponential interarrival times with a mean of 1 minute. This is followed by a "Queue" block labeled "Machine Queue" (Fig. 3.12), which stores the parts while they are waiting for processing. This queue has infinite capacity by default and merges the parts from the Create block with those parts that need to be reworked after inspection.

Following the Machine Queue block is an "Activity" block labeled "Machine Part." In the dialog box for this latter block (Fig. 3.13), we specify that one part can be processed at a time. We also select "Uniform, Real" as the processing-time distribution and then set its minimum and maximum values to 0.65 and 0.70 minute, respectively. This Activity block is connected to a second Queue block labeled "Inspect Queue," where parts wait for the inspection process. The output of this Queue block is connected to a second Activity block labeled "Inspect Part," where inspection times are uniformly distributed between 0.75 and 0.80 minute.

The Activity block corresponding to the inspector is connected to the "Select Item Out" block labeled "Random Output," which is used to determine whether a part is good or bad. In its dialog box (Fig. 3.14), we specify that parts will leave randomly through the block's outputs. In the table we enter the probabilities

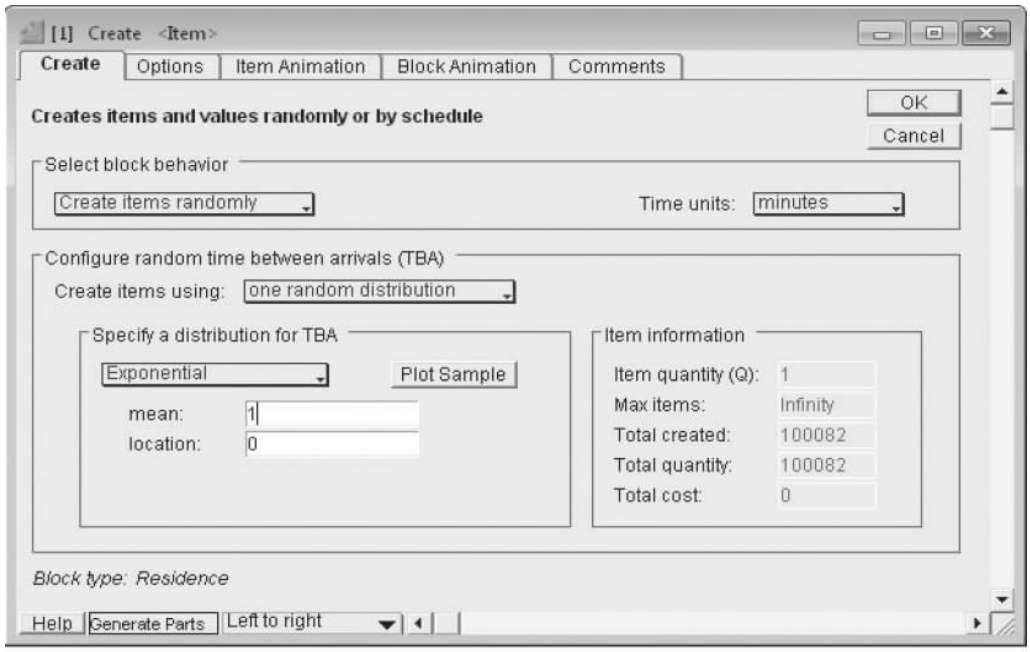

FIGURE 3.11
Dialog box for the ExtendSim Create block "Generate Parts."

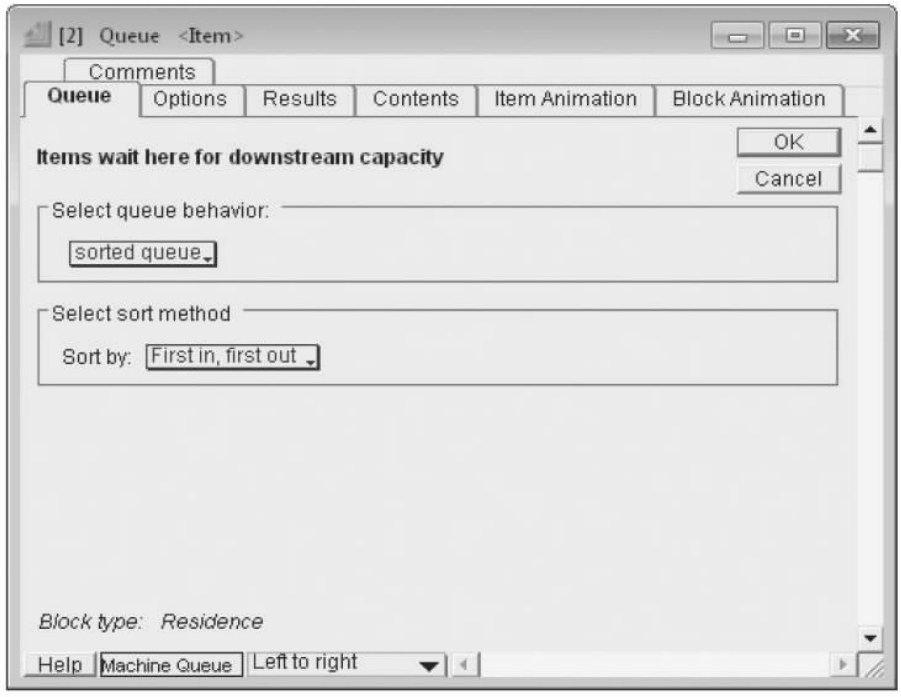

FIGURE 3.12
Dialog box for the ExtendSim Queue block "Machine Queue."

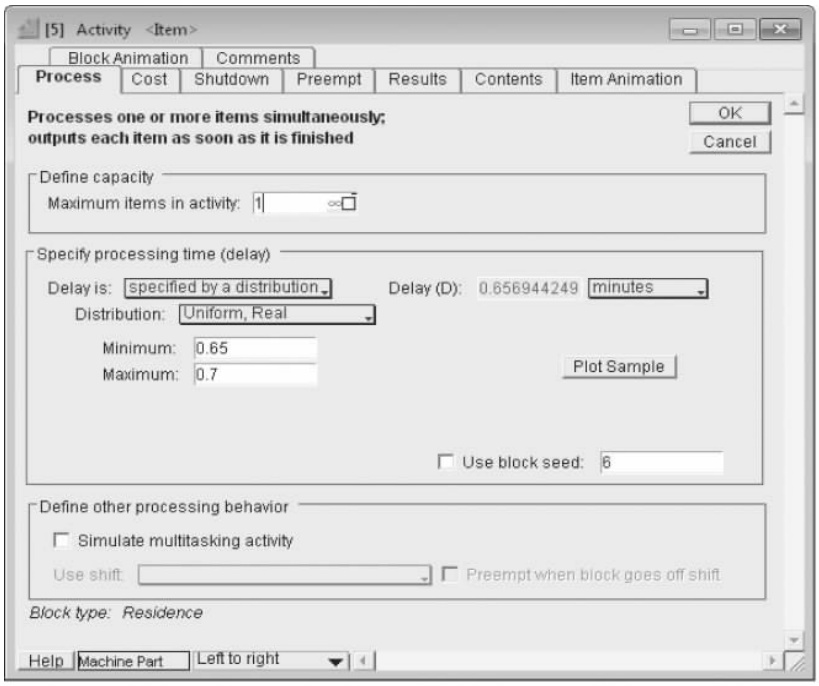

FIGURE 3.13
Dialog box for the ExtendSim Activity block "Machine Part."

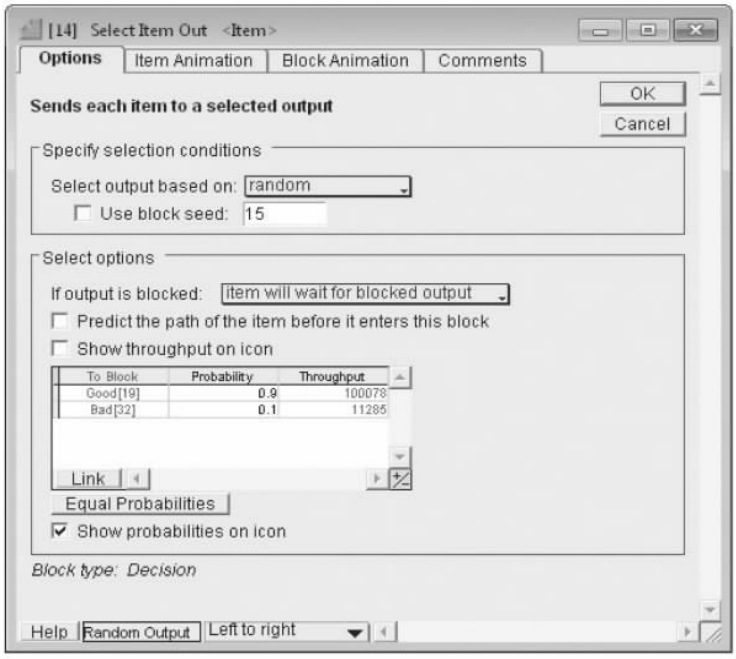

FIGURE 3.14
Dialog box for the ExtendSim Select Item Out block "Random Output."

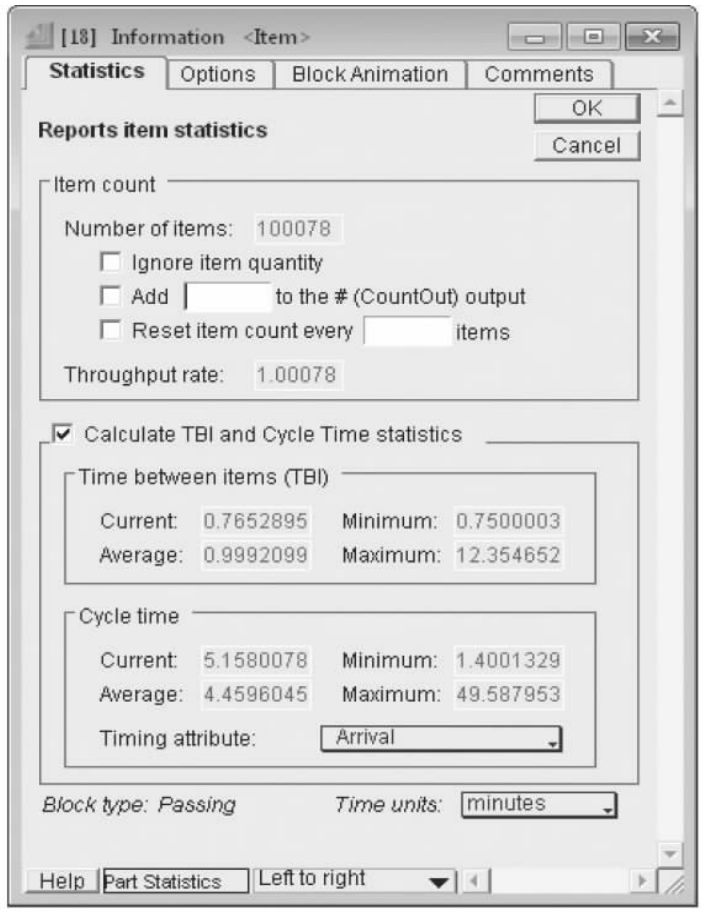

FIGURE 3.15
Dialog box for the ExtendSim Information block "Part Statistics."

0.9 and 0.1, indicating that 90 percent of the parts will be sent through the top output as "Good" and 10 percent of the parts will be sent through the lower output as "Bad." We also choose to have the probabilities displayed on the output connections of this block.

The next block in the model is an "Information" block labeled "Part Statistics," which computes output statistics for completed parts. In its dialog box (Fig. 3.15), we see that 100,078 (good) parts were completed and that the average time in system (cycle time) was 4.46 minutes. The last block in the model is an "Exit" block labeled "Destroy Parts" (see Fig. 3.10), where the completed parts are removed from the model.

The time units for the model (minutes), the simulation run length (100,000), and the desired number of runs (1) are specified in the "Simulation Setup" option that is accessed from the "Run" pull-down menu (not shown) at the top of the screen. The Notebook for the model (Fig. 3.16), which is accessed from the

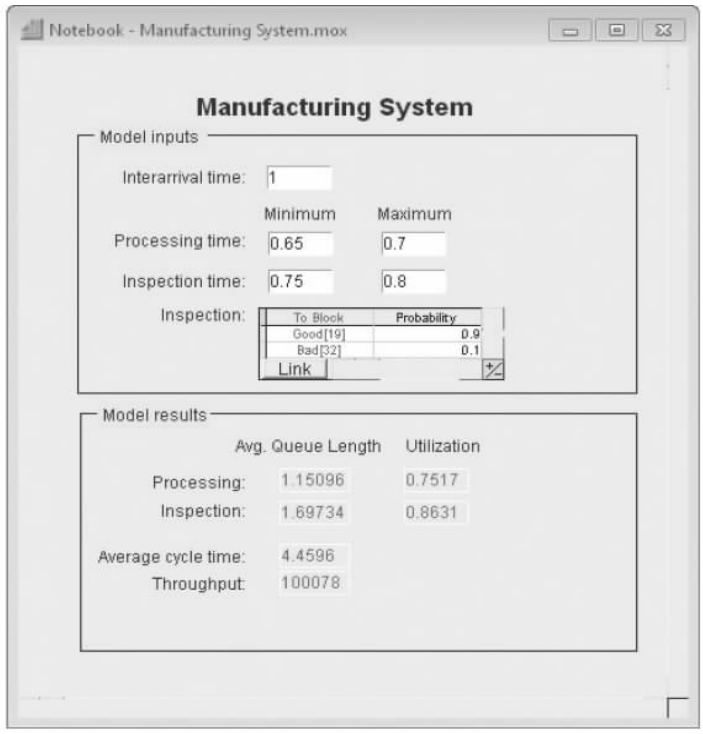

FIGURE 3.16
ExtendSim Notebook for the manufacturing system.

"Window" pull-down menu, brings together important input parameters and results for the model.

In Fig. 3.17 we give a version of the ExtendSim model that uses hierarchy (see Sec. 3.4.1). If we double-click on the hierarchical block named "Process" (at the first level of hierarchy), then we go down to the second level of hierarchy where we see the original Machine Queue and Machine Part blocks, as shown in Fig. 3.18.

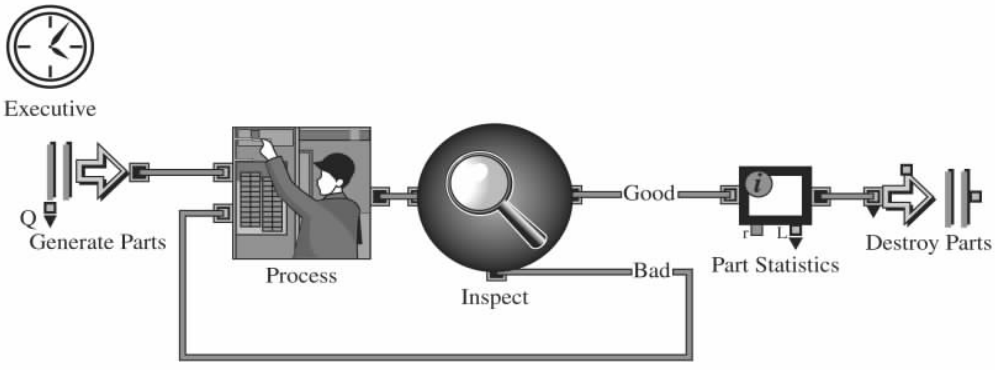

FIGURE 3.17
Hierarchical ExtendSim model for the manufacturing system.

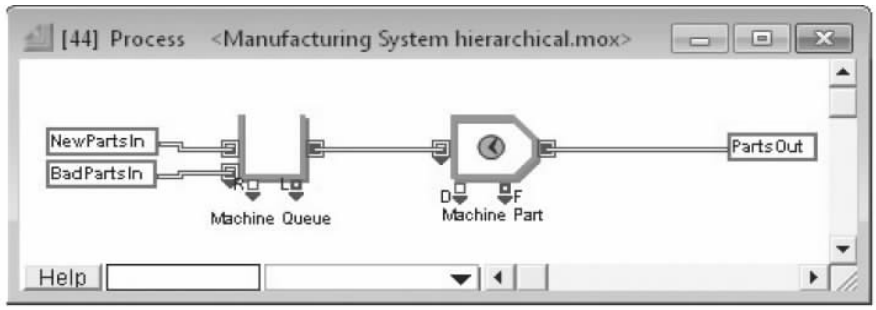

FIGURE 3.18
Components of the Process hierarchical block.

3.5.3 Simio

Simio [Simio (2013) and Kelton et al. (2011)] is an object-oriented (see Sec. 3.6) suite of simulation and scheduling products marketed by Simio LLC (Sewickley, Pennsylvania). Simio is a *sim*ulation-modeling framework based on *i*ntelligent *o*bjects, which allows one to build models using either the default Standard Library (for discrete-event simulation) or by graphically creating entirely new objects. (An object in Simio has properties, states, and logic.) The Standard library, which contains 15 object definitions, can be modified and extended using process logic (see below), and new objects can be stored in libraries for use in other simulation projects.

An object in a library might be a customer, machine, doctor, or anything else that you might find in a system. A model is constructed in Simio by dragging objects into the "Facility" window, connecting them by links to indicate the flow of entities through the simulated system, and then detailing the objects by using a property editor. The model logic and animation are built in a single step, typically in a two-dimensional view for ease of modeling. However, one can switch to a three-dimensional (3-D) perspective view with just a single keystroke.

Building an object in Simio is identical to building a model, since there is no difference between the two constructs. Whenever you build a model, it is by definition an object that can be used in another model. For example, if you combine two machines and a robot into a model of a workstation, then the workstation model is itself an object that can then be used in other models. Every model that is built in Simio is automatically a building block that can be used in constructing hierarchical models.

When you instantiate an object into a model, you may specify "properties" (static input parameters) of the object that govern the behavior of this specific instance of the object. For example, a property of a machine might be its processing time. The developer of an object decides on the number of properties and their meanings. Properties in Simio can be numerical values, Boolean variables, text strings, etc.

In addition to properties, objects have "states" that change values as a result of the execution of the object's logic. A state for a machine might be its status (e.g., idle or busy). Properties and states together constitute the attributes of an object.

An object in Simio may be defined from one of five base classes, which provides the underlying behavior for the object. The first class is the "fixed object,"

which has a fixed location in the model and is used to represent something in a system that does not move from one location to another, such as a machine in a factory or an operating room in a hospital.

An "entity" is an object that can move through 3-D space over a network of links and nodes. Examples of entities are parts in a manufacturing system, and patients, nurses, and doctors in a hospital. Note that in traditional simulation packages entities are passive and are acted upon by the model processes (see Sec. 3.3.2). However, in Simio the entities are intelligent objects that can control their own behavior.

"Link" and "node" objects are used to build networks over which entities may flow. A link defines a pathway for entities to move from one object to another, whereas a node defines the beginning or ending point of a link. Links and nodes can be combined together into complex networks. A link could be an escalator with a fixed travel time or it could represent a conveyor.

The final class of objects is a "transporter," which is a subclass of the entity class. A transporter is an entity that has the added capabilities to pick up, carry, and drop off one or more other entities. A transporter could be used to model a bus, a forklift truck, or any other object that has the ability to carry other entities from one location to another.

A key feature of Simio is the ability to create a wide range of object behaviors from the base classes. The Simio modeling framework is application-domain neutral, i.e., these base classes are not specific to a particular application area such as manufacturing or health care. However, it is easy to build application-oriented libraries composed of intelligent objects from the base classes. Simio's design philosophy dictates that domain-specific logic belongs in the objects built by users, and it is not programmed into the core system.

The process approach (see Sec. 3.3.2) is commonly used for extending an object's logic or for building new objects. A process is defined in Simio using a flowchart, where each step in the flowchart defines some action to perform. There are over 50 different process steps available in Simio to perform specific actions such as delay by time, wait to seize a resource, etc. Process logic can be inserted into a specific instance of an object to modify or extend its behaviors. For example, an object representing a machine might use process logic to seize and hold a repairman during a breakdown.

There are an essentially unlimited number of random-number streams available in Simio. Furthermore, the user has access to 19 standard theoretical probability distributions and to empirical distributions. There is an easy mechanism for making independent replications of a simulation model and for obtaining point estimates and confidence intervals for performance measures of interest. A number of plots are available such as time plots, histograms, bar charts, and pie charts.

Simio provides a 3-D interactive environment for building and running simulation models, which is useful for accurately modeling spatial relationships and for communicating model behavior to the simulation project's stakeholders. However, Simio also provides a set of sophisticated features for performing and analyzing simulation experiments. In particular, a model may have an associated "experiment" that specifies a set of scenarios to execute. Each scenario *may* have one or more input controls and will have one or more output responses. The input controls are factors that are changed from one scenario to the next (e.g., the number of machines in a workstation), and the output responses are the measures of performance (e.g., average

time in system of a part) that are used to evaluate the efficacy of the different scenarios. Furthermore, each scenario can be replicated a specified number of times and these replications can be simultaneously executed across multiple processor cores or across different computers on a network, which will greatly reduce the time required for experimentation. Simio's built-in analysis tools include a procedure for automatically selecting the best scenario from a set of candidate scenarios [see Sec. 10.4.3 and Kim and Nelson (2001)] and SMORE plots [see Nelson (2008)]. A SMORE plot simultaneously displays a point estimate and confidence interval for the expected value of a response, as well as a superimposed box plot (see Sec. 6.4.3). The "OptQuest for Simio" (see Sec. 12.5.2) optimization module is available as an option.

Although Simio is primarily oriented toward performing discrete-event simulation using an object-oriented approach, Simio also supports modeling continuous-flow systems, performing agent-based simulation (because of its object orientation), and performing discrete-event simulation using the process approach. Moreover, Simio can also be used in an operational setting as a risk-based planning and scheduling tool to improve the day-to-day functioning of an organization.

We now develop a Simio model of the simple manufacturing system discussed in Sec. 3.5.1. The Simio model for this system is shown in Figure 3.19 and is composed of a "Source" object named "Part_Arrivals" that creates the jobs arriving to the system, a "Server" object named "Machine_Part" that models the machining operation, a Server object named "Inspect_Part" that models the inspection process, and a "Sink" object named "Part_Departures," where entities leave the system. In this example, we use a zero-time link called a "Connector" to define the travel paths between the Source, Servers, and Sink objects.

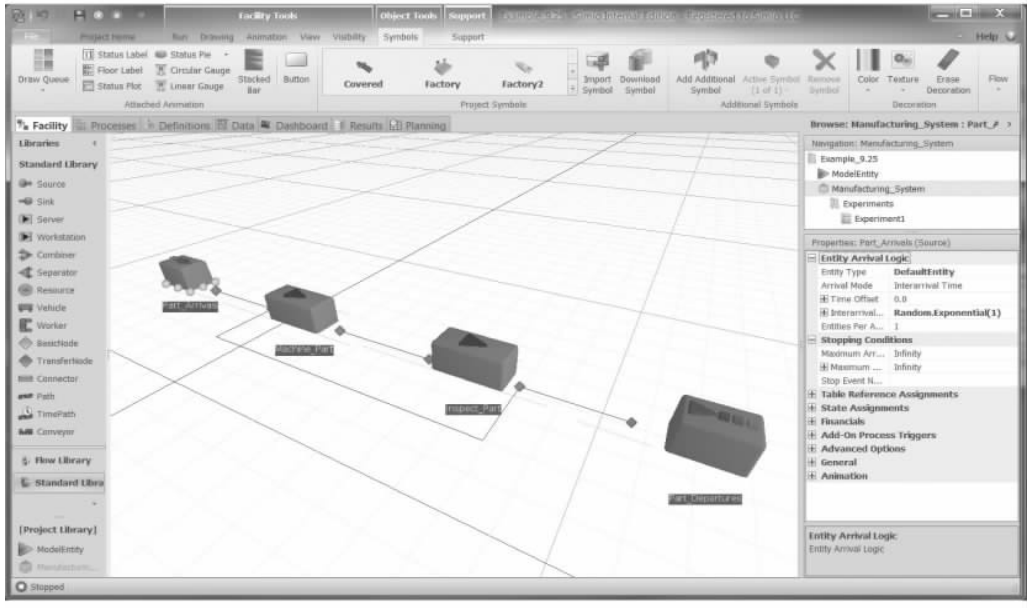

FIGURE 3.19
Simio model for the manufacturing system.

The small circular "selection handles" surrounding the Part_Arrivals object indicate that the object was selected for editing (by clicking on it with the mouse). The properties of a selected object are edited using the "Property Editor," which is on the lower-right side of the screen. The Part_Arrivals object is used to generate arrivals to the system based on an "Arrival Mode." The default "Interarrival Time" mode used in this example specifies that the distribution of interarrival times is exponential with a mean of 1 minute. [Alternatively, the "Time Varying Arrival Rate" mode generates arrivals in accordance with a nonstationary Poisson process (see Sec. 6.12.2) and the "Arrival Table" mode schedules arrivals using data stored in a table or an external source such as a spreadsheet.]

Figure 3.20 displays the properties for the Machine_Part object. The properties are organized into categories that can be expanded and collapsed with the +/− signs to the left of the category name. These properties specify that "Processing Time" is uniformly distributed on the interval [0.65, 0.70] minute. Note that this expression can be typed in directly or specified using an "Expression Editor," which can be accessed using a pull-down arrow on the right side of the field (not shown). If failures of Machine_Part

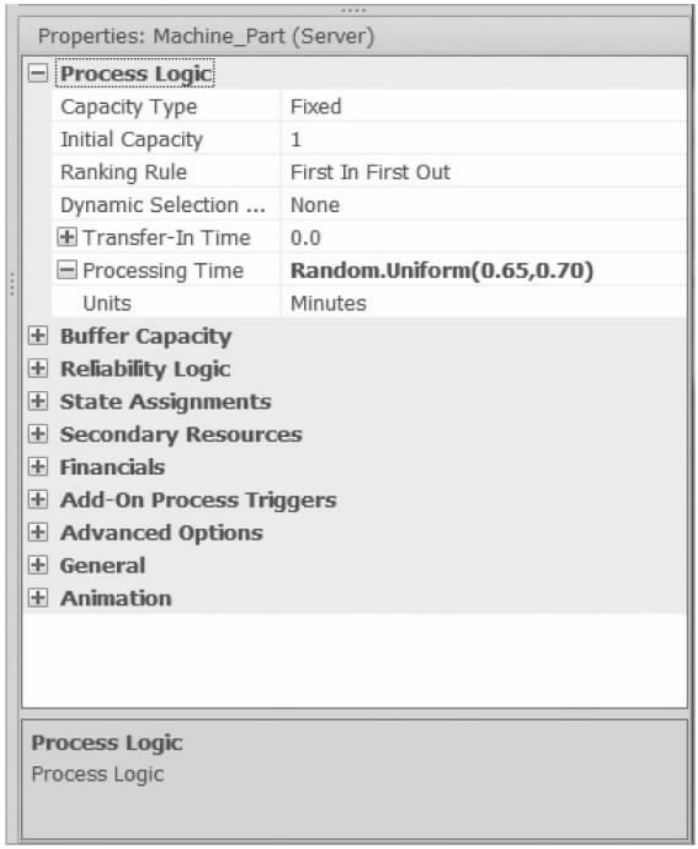

FIGURE 3.20
Properties of the Machine_Part object.

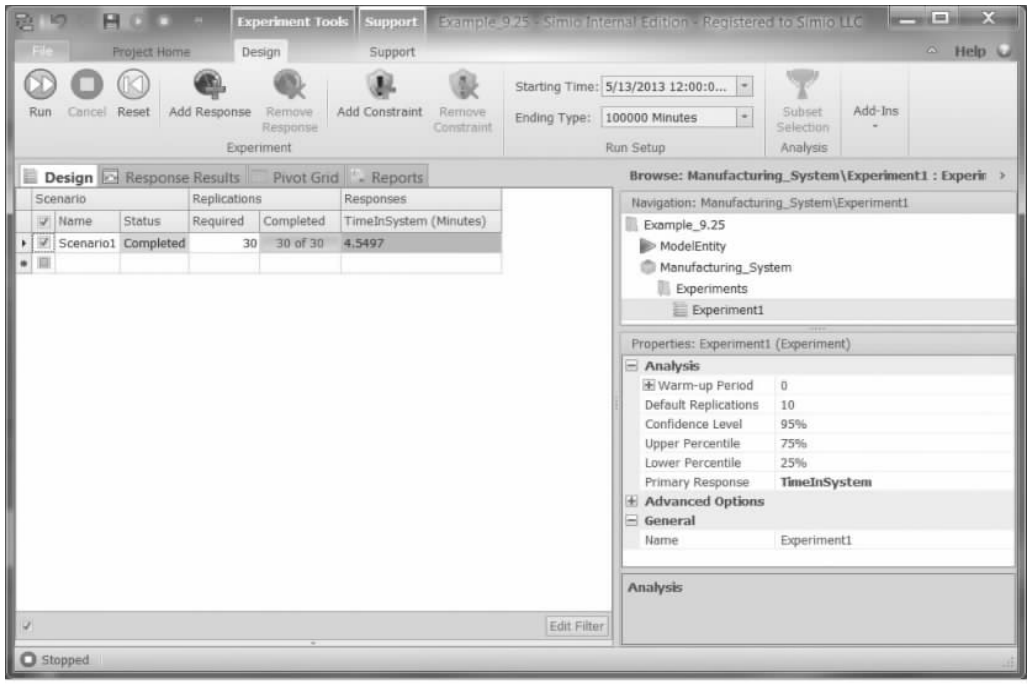

FIGURE 3.21
Design view for specifying an experiment.

were desired, then they would be specified under the "Reliability Logic" category. The "Financials" category can be used to specify usage rates for activity-based costing.

We do not show the Property Editor for the Inspect_Part object, where the inspection times are specified to be uniformly distributed on the interval [0.75, 0.80] minute. The two connectors leaving Inspect_Part (see Fig. 3.19) have link weights of 0.9 and 0.1, respectively, and use a routing rule on its exit node that is based on "By Link Weight."

Figure 3.21 shows the specification of and partial results from a simple experiment for our model, which says to make 30 replications of the simulation and to observe the average time in system of a part for a run length of 100,000 minutes. Note that the average time in system over the 30 replications was 4.55 minutes.

These same results are shown in Figure 3.22 in the form of a SMORE plot. This plot shows a point estimate ("dot") and a 95 percent confidence interval ("small" shaded rectangle over the dot) for the expected average time in system. Superimposed over this is a box plot showing the minimum, 25th percentile, median, 75th percentile, and maximum of the 30 observed values of average time in system. Finally, the two outer shaded rectangles are 95 percent confidence intervals for the 25th and 75th percentiles.

A standard report that is automatically produced by a simulation model can potentially contain a large amount of output statistics, which can make it difficult to find the information that is really of interest. To help alleviate this problem, Simio presents the simulation results in the form of a "Pivot Grid" (similar to a pivot table

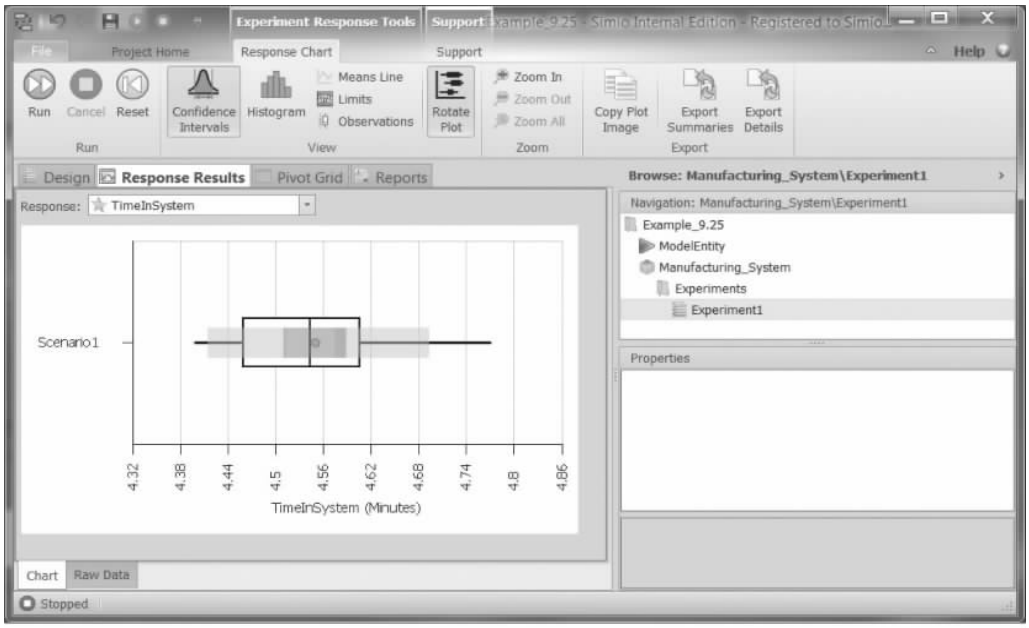

FIGURE 3.22
SMORE plot for average time in system.

in Excel), which can easily be customized to display the statistics of interest in an appropriate format. A Pivot Grid for the simulation results produced by the 30 replications is shown in Figure 3.23. Note that the machine and inspector had a utilization of 0.75 and 0.86, respectfully.

Average | Minimum | Maximum | Half Width

Object Type	Object Name	Data Source	Category	Data Item	Statistic	Scenario1 Average	Scenario1 Minimum	Scenario1 Maximum	Scenario1 Half Width
ModelEntity	DefaultEntity	[Population]	Content	NumberInSystem	Average	4.5548	4.3925	4.7878	0.0436
					Maximum	33.5000	27.0000	43.0000	1.6370
			FlowTime	TimeInSystem	Average (Minutes)	4.5497	4.3988	4.7710	0.0393
					Maximum (Minutes)	69.2576	45.7114	90.1023	3.5776
					Minimum (Minutes)	1.4005	1.4001	1.4012	0.0001
Server	Machine_Part	[Resource]	Capacity	UnitsUtilized	Average	0.7508	0.7465	0.7574	0.0009
					Maximum	1.0000	1.0000	1.0000	0.0000
		InputBuffer	Content	NumberInStation	Average	1.1666	1.1284	1.2200	0.0098
					Maximum	19.1333	15.0000	26.0000	0.8589
			HoldingTime	TimeInStation	Average (Minutes)	1.0487	1.0204	1.0922	0.0078
					Maximum (Minutes)	12.5002	9.8503	17.3396	0.5746
					Minimum (Minutes)	0.0000	0.0000	0.0000	0.0000
	Inspect_Part	[Resource]	Capacity	UnitsUtilized	Average	0.8620	0.8571	0.8697	0.0011
					Maximum	1.0000	1.0000	1.0000	0.0000
		InputBuffer	Content	NumberInStation	Average	1.7754	1.6449	1.9691	0.0338
					Maximum	26.2667	20.0000	35.0000	1.4308
			HoldingTime	TimeInStation	Average (Minutes)	1.5959	1.4820	1.7654	0.0290
					Maximum (Minutes)	19.9031	14.6897	26.8397	1.1073
					Minimum (Minutes)	0.0000	0.0000	0.0000	0.0000
Sink	Part_Departures	[DestroyedObjects]	FlowTime	TimeInSystem	Average (Minutes)	4.5497	4.3988	4.7710	0.0393
					Maximum (Minutes)	69.2576	45.7114	90.1023	3.5776
					Minimum (Minutes)	1.4005	1.4001	1.4012	0.0001
					Observations	100,102.6333	99,482.0000	100,878.0000	118.1014

FIGURE 3.23
Simulation results displayed in a Pivot Grid.

3.5.4 Other General-Purpose Simulation Packages

There are several other well-known, general-purpose simulation packages, including AnyLogic [AnyLogic (2013)], SIMUL8 [SIMUL8 (2013)], and SLX [Wolverine (2013)].

3.6 OBJECT-ORIENTED SIMULATION

In the last 20 years there has been a lot of interest in object-oriented simulation [see, e.g., Joines and Roberts (1998) and Levasseur (1996)]. This is probably an outgrowth of the strong interest in object-oriented programming in general. Actually, both object-oriented simulation and programming originated from the object-oriented simulation language SIMULA, which was introduced in the 1960s.

In *object-oriented simulation* a simulated system is considered to consist of *objects* (e.g., an entity or a server) that interact with each other as the simulation evolves through time. There may be several instances of certain object types (e.g., entities) present concurrently during the execution of a simulation. Objects contain *data* and have *methods* (see Example 3.1). Data describe the state of an object at a particular point in time, while methods describe the actions that the object is capable of performing. The data for a particular object instance can only be *changed* by its own methods. Other object instances (of the same or of different types) can only *view* its data. This is called *encapsulation*.

Examples of true object-oriented simulation packages are AnyLogic, FlexSim, and Simio. Three major features of such a simulation package are inheritance, polymorphism, and encapsulation (defined above). *Inheritance* means that if one defines a new object type (sometimes called a *child*) in terms of an existing object type (the *parent*), then the child type "inherits" all the characteristics of the parent type. Optionally, certain characteristics of the child can be changed or new ones added. *Polymorphism* is when different object types with the same ancestry can have methods with the same name, but when invoked may cause different behavior in the various objects. [See Levasseur (1996) for examples of inheritance and polymorphism.]

> **EXAMPLE 3.1.** In a manufacturing system, the fabrication area and the assembly area might be considered as objects (first level of hierarchy). In turn, the fabrication area might consist of machine, worker, and forklift-truck objects (second level of hierarchy). Data for a forklift might include its speed and the maximum weight that it can lift. A method for a forklift might be the dispatching rule that it uses to choose the next job.

Some vendors claim that their simulation software is object-oriented, but in some cases the software may not include inheritance, polymorphism, or encapsulation. Furthermore, certain of the above three features are sometimes assigned different meanings.

The following are possible advantages of object-oriented simulation:

- It promotes code reusability because existing objects can be reused or easily modified.
- It helps manage complexity by breaking the system into different objects.
- It makes model changes easier when a parent object can be modified and its children objects realize the modifications.
- It facilitates large projects with several programmers.

Possible disadvantages of the object-oriented approach are:

- Some object-oriented simulation packages may have a steep learning curve.
- One must do many projects and reuse objects to achieve its full benefits.

3.7 EXAMPLES OF APPLICATION-ORIENTED SIMULATION PACKAGES

In this section we list some of the application-oriented simulation packages that are currently available.

Manufacturing. AutoMod [Applied Materials (2013)], Enterprise Dynamics [INCONTROL (2013)], FlexSim [FlexSim (2013)], Plant Simulation [Siemens (2013)], ProModel [ProModel (2013)], and WITNESS [Lanner (2013)] (see Sec. 14.3 for further discussion).

Communications networks. OPNET Modeler [Riverbed (2013)] and QualNet [SCALABLE (2013)].

Health care. FlexSim Healthcare [FlexSim (2013)] and MedModel [ProModel (2013)].

Process reengineering and services. Process Simulator [ProModel (2013)], ProcessModel [ProcessModel (2013)], and ServiceModel [ProModel (2013)].

Animation (*stand-alone*). Proof Animation [Wolverine (2013)].