original

# CHAPTER 3

# SIMULATION SOFTWARE

Recommended sections for a first reading: 3.1 through 3.4

## 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, random values from the U(0, 1) probability distribution
- Generating random values 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 and analyzing data
- 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 languages. Furthermore, we believe that the improvement, standardization, and greater availability of these languages has been one of the major factors in the increased popularity of simulation in recent years.

We discuss in Sec. 3.2 the relative merits of using a simulation language rather than a general-purpose language such as FORTRAN or C for programming simulation models. Most simulation languages in use today employ one of two modeling approaches or orientations. These two orientations, called the *event-scheduling* and the *process* approaches, are discussed in Sec. 3.3. Desirable features for simulation software, including animation, are described in Sec. 3.4. In Secs. 3.5 through 3.8 we present brief descriptions of GPSS, SIMAN, SIMSCRIPT II.5, and SLAM II, which are probably the most widely used simulation languages in the United States. A simulation model of the $M/M/1$ queue (see Sec. 1.4.3) is also given in each language. These languages are compared in Sec. 3.9, followed by a discussion of other simulation software (e.g., application-oriented simulators) in Sec. 3.10.

## 3.2 COMPARISON OF SIMULATION LANGUAGES WITH GENERAL-PURPOSE LANGUAGES

One of the most important decisions a modeler or analyst must make in performing a simulation study is the choice of a language. An inappropriate choice may in itself cause a simulation project to be unsuccessful if it cannot be completed on time. The following are some advantages of programming a simulation model in a simulation language rather than in a general-purpose language, e.g., FORTRAN, C, Pascal, or BASIC:

- Simulation languages automatically provide most of the features needed in programming a simulation model (see Secs. 3.1 and 3.4), resulting in a significant decrease in programming time.
- They provide a natural framework for simulation modeling. Their basic building blocks are more closely akin to simulation than are those in a language like FORTRAN.
- Simulation models are generally easier to change when written in a simulation language.
- Most simulation languages provide dynamic storage allocation during execution.
- They provide better error detection because many potential types of errors have been identified and are checked for automatically. Since fewer lines of code have to be written, the chance of making an error will probably be smaller. (Conversely, errors in a new version of a simulation language itself may be difficult for a user to find.)

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

- Most modelers already know a general-purpose language, but this is often not the case with a simulation language.
- FORTRAN or BASIC is available on virtually every computer, but a particular simulation language may not be accessible on the computer that the analyst wants to use.
- An *efficiently* written FORTRAN or C program may require less execution time than the corresponding program written in a simulation language. This is because a simulation language is designed to model a wide variety of systems with one set of building blocks, whereas a FORTRAN program can be tailored to the particular application. This consideration has, however, become less important with the availability of relatively inexpensive, high-speed microcomputers and engineering work stations.
- General-purpose languages may allow greater programming flexibility than certain simulation languages.
- Software cost may be lower (but not necessarily project cost).

Although there are clear advantages to using both types of languages, we believe, in general, that a modeler would be prudent to give serious consideration to using a simulation language. If such a decision has indeed been made, the criteria discussed in Secs. 3.4 and 3.9 may be helpful in deciding which particular simulation language to choose.

## 3.3 CLASSIFICATION OF SIMULATION SOFTWARE

In this section we discuss various aspects of simulation software, including two different ways in which it can be classified.

### 3.3.1 Simulation Languages vs. Simulators

There are currently two major classes of simulation software: languages and simulators. A *simulation language* is a computer package that is general in nature but may have special features for certain types of applications. For example, SIMAN and SLAM II have manufacturing modules for conveyors and automated guided vehicles. A model is developed in a simulation language by writing a program using the language's modeling constructs. The major strength of most languages is their ability to model almost any kind of system, regardless of the system's operating procedures or control logic. Possible drawbacks of simulation languages are the need for programming expertise and the possibly long coding and debugging time associated with modeling complex systems (relative to simulators, if applicable).

A *simulator* is a computer package that allows one to simulate a system contained in a *specific* class of systems with little or no programming. For example, there are currently simulators available for certain types of manufacturing, computer, and communication systems. The particular system of interest (in the domain of the package) is typically selected for simulation by the use of menus and graphics, without the need for programming. The major advantage of a simulator is that "program" development time may be considerably less than that for a simulation language. This may be very important given the tight time constraints in many business environments. Another advantage is that most simulators have modeling constructs related specifically to the components of the target class of systems, which is particularly desirable for operational personnel. Also, people without programming experience or who use simulation only occasionally (e.g., a manufacturing engineer in a factory) often prefer simulators because of their ease of use. The major drawback of many simulators is that they are limited to modeling only those system configurations allowed by their standard features. This difficulty can be somewhat overcome if the simulator has "programming-like" commands to model complex decision logic; most of the model would still be developed using menus and graphics. (This capability might be available in the simulator itself or in external routines called by the simulator.) Simulators are currently most often used for *high-level analyses*, where the system is modeled at an aggregate level without including details of the control logic.

### 3.3.2 Modeling Approaches

Almost all simulation languages use one of two basic approaches to discrete-event simulation modeling; these approaches are also used by modelers using a general-purpose language. In the *event-scheduling approach*, used in the programs in Chaps. 1 and 2, 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. The event-scheduling approach is available in SIMAN, SIMSCRIPT II.5, and SLAM II.

A *process* is a time-ordered sequence of interrelated events separated by passages 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 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

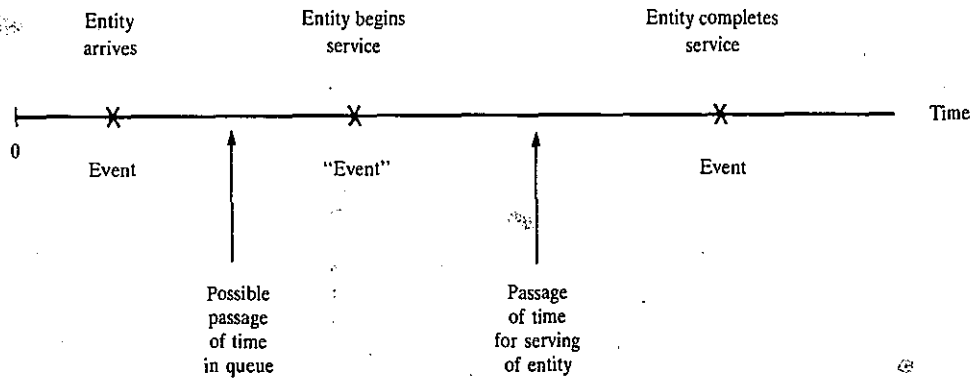



**FIGURE 3.1**
Process describing the flow of an entity through a system.

single-server queueing system. (This process routine describes the entire experience of a customer as it progresses 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 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 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

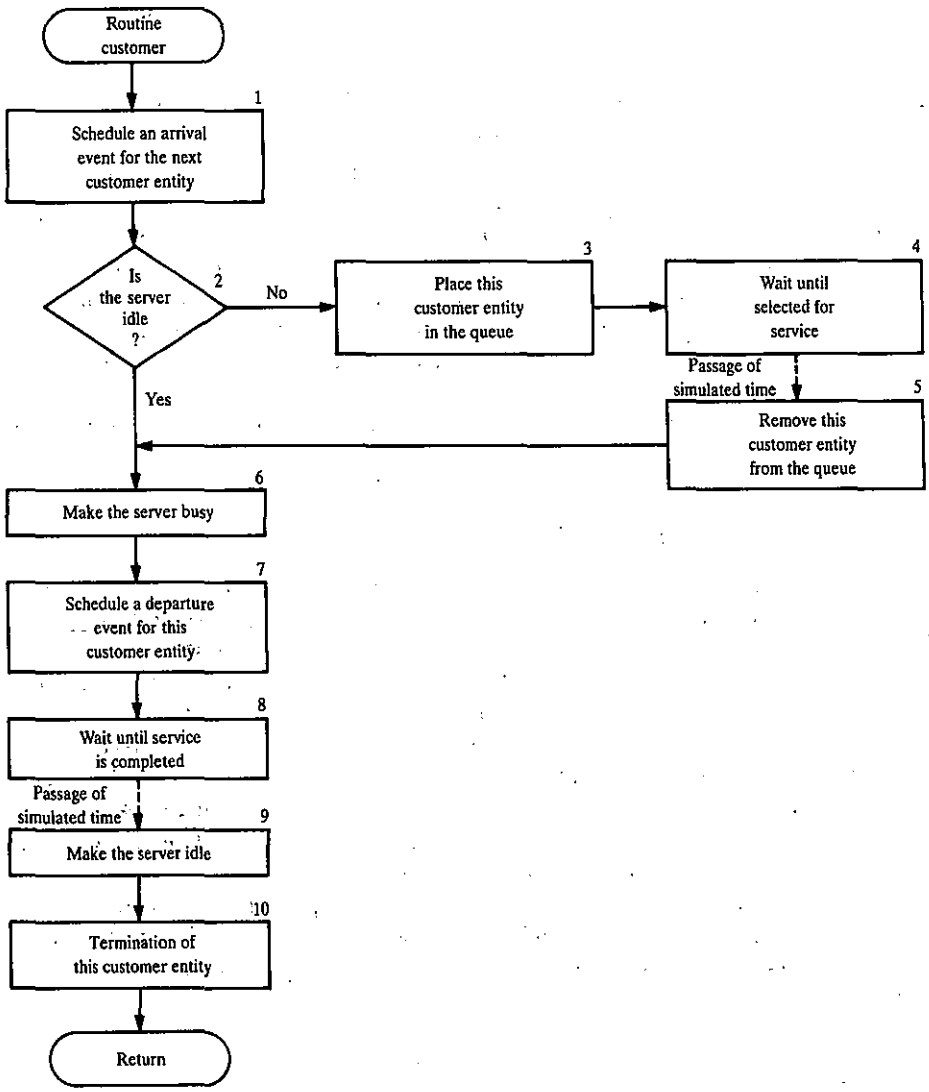



**FIGURE 3.2**
Prototype customer-process routine for a single-server queueing system.

service, it 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. [A more detailed explanation of the process approach in the context of SIMSCRIPT II.5 may be found in Law and Larmey (1984).]

A simulation using the process approach also evolves over time by executing the events in increasing order of their time of occurrence. Internally, the two approaches to simulation are very similar (e.g., both approaches use a

simulation clock, an event list, a timing routine, etc.). They differ mainly in the language constructs that they make available to model a system. Process statements are more "macro" in nature and automatically translate certain situations commonly occurring in a simulation model, e.g., customers arriving to a queueing system, into the corresponding event logic.

The process approach has several advantages over the event-scheduling approach. For many types of systems the process approach is more natural in some sense, since one process routine describes the entire experience of the corresponding process entity. Furthermore, a process simulation model of a system usually requires fewer lines of code than the comparable program using the event-scheduling approach. On the other hand, the process approach as implemented in some simulation languages is less flexible than the event-scheduling approach.

The process approach is the major modeling orientation in GPSS/H, GPSS/PC, SIMAN, SIMSCRIPT II.5, and SLAM II.

### 3.3.3 Common Modeling Elements

There are a number of modeling elements common to the simulation packages (languages or simulators) discussed in this chapter. An *entity* (or *transaction*) is a person or object that arrives to a system, is "serviced" in some manner, and then usually departs. Examples of entities are a customer arriving to a barbershop, a part in a factory, and a message for a communication system. An *attribute* (or *parameter*) is a piece of information that describes or characterizes an entity, such as haircut type for a customer, due date for a part, or the length of a message. A *queue* (or *file* or *set*) is a collection of entities with some common characteristic, such as parts waiting to be processed on a machine. Entities in a queue may be processed in a FIFO or LIFO manner, or based on the value of some entity attribute. A *resource* is a person or "machine" that provides service to an entity while it is present in a system. Examples are a barber, a worker or machine in a factory, and a node or link in a communication system.

## 3.4 DESIRABLE SOFTWARE FEATURES

In Sec. 3.1 we discussed some basic features or capabilities needed in programming a simulation model. We now continue this discussion by presenting a number of additional features that should be available in a contemporary simulation package, with these features being grouped into five categories. [See Law and Haider (1989), and also the discussion of material-handling modules in Sec. 13.3.]

### 3.4.1 General Features

Perhaps the most important feature for a simulation package to have is *modeling flexibility*, because no two systems are exactly the same. If the

simulation package does not have the necessary capabilities for a particular application then the system must be approximated, resulting in a model with unknown validity. Entities should have *general attributes* (e.g., due date, message length, etc.), which can be appropriately changed; this capability is generally available in simulation languages but is less common in simulators.

*Ease of model development* is another very important feature, due to the short time frame for many projects. The accuracy and speed of the modeling process will be increased if the package has good *debugging aids*, such as an interactive debugger, on-line input error checking, and on-line help.

*Fast model execution speed* is particularly important for very large models (e.g., certain military applications) and when the simulation model is to be run on a microcomputer. For a complicated simulation model of a 40-machine food-packaging plant, it took 7 hours to simulate 2 weeks of production on a 16-megahertz microcomputer.

The *maximum model size* allowed by the simulation package may be an important factor when the model is to be executed on a microcomputer. For some packages, the maximum model size is currently less than 100 K bytes. This potential difficulty will become less important as many vendors are beginning to offer versions with extended model sizes.

It is also desirable for a simulation package to be available for a number of different computer classes (i.e., microcomputer, work station, and minicomputer/mainframe), and for the software to be *compatible across these classes*. Thus, for example, a model could be developed on a microcomputer and then uploaded to a minicomputer or mainframe for execution of the production runs.

Finally, in some applications (e.g., steel manufacturing) it is convenient for the software to have capabilities for *combined discrete-continuous simulation* (see Sec. 1.8.2).

### 3.4.2 Animation

Easy-to-use animation is one of the main reasons for the increased popularity of simulation modeling. In an *animation*, key elements of a system (e.g., machines and parts) are represented on a CRT by icons that change shape, color, or position when there is a change of state in the simulation. Thus, a system can be seen graphically to change over time. Most contemporary animation packages operate in a *concurrent mode*, where the animation is displayed while the simulation is actually running (perhaps slowed down to allow for visual comprehension). On the other hand, some animation packages function in a *playback mode*, where the animation is displayed after the simulation is completed from state changes recorded in a disk file. Several examples of animation and graphics are given in color Plate 1.

The major reason for the popularity of animation is its ability to communicate the essence of a simulation model (or of simulation itself) to managers and other key project personnel, greatly increasing the model's credibility. Other potential benefits of animation are:

- Debugging a simulation computer program
- Showing that a simulation model is *not* valid
- Suggesting improved operational procedures or control logic for a system
- Understanding the dynamic behavior of a system
- Training operational personnel

Animation also has certain shortcomings or disadvantages. In particular, it is not a substitute for a careful statistical analysis of the simulation output data. One cannot conclude that a system is "well defined" by watching an animation for a "short" period of time since, if the simulation were run for a longer period of time, a crucial piece of equipment might fail and cause a major system bottleneck. Animating a simulation model increases model development time, and simulation packages with an animation capability are often considerably more expensive. Finally, only part of a simulation model's logic can actually be seen in an animation; thus, a "correct" animation is no guarantee of a valid or debugged model.

There are a number of desirable features for an animation package. First and foremost, since animation is primarily a communication tool, it is important for it to look realistic (particularly for presentations to high-level managers). The user should be able to *create high-resolution icons* using bit-mapped rather than character graphics. There should be *smooth movement of icons* across the computer screen, rather than "jumpy" or "pulsating" movement. It should be possible to *store icons in a library* for use in a future model, and the library should come with *standard icons* to facilitate animation development. The animation should be *easy to develop*, relying more on menus and graphics that on programming. There should be the capability for *multiple-screen layout*, since some models will not "fit" on a single standard computer screen. Additional animation features are discussed in Law and Haider (1989).

A useful graphical companion to animation is dynamic presentation-quality graphics, where histograms, level meters, dials, etc., are updated as the simulation progresses through time.

### 3.4.3 Statistical Capabilities

Since most real-world systems exhibit some sort of random behavior, a simulation package must contain good statistical capabilities that should actually be used. In general, each source of system randomness (interarrival times, service times, machine operating times, etc.) needs to be modeled by a probability distribution, *not* just its mean (see Sec. 4.7). A simulation package should contain a wide variety of *standard distributions* (e.g., exponential, gamma, and triangular), should be able to use *distributions based on observed system data* (see Sec. 6.2.4), and should contain a *multiple-stream random-number generator* to facilitate comparing alternative system designs (see Secs. 7.1 and 11.2).

Since random samples from the input probability distributions "drive" a simulation model through time, simulation output data (e.g., daily throughputs in a factory) are also random and appropriate statistical techniques must be used to design and interpret the simulation runs. A simulation package should contain a single *command* to make *independent replications* of the model automatically, with each replication using different random numbers, starting in the same initial state, and resetting the statistical counters to zero. We should be able to specify a *warmup period* (at the end of which statistical counters are reset to zero) and to construct *confidence intervals* for desired measures of performance (e.g., mean daily throughput) in order to determine the statistical precision of the simulation results.

### 3.4.4 Customer Support

Most users of simulation software require some level of ongoing support from the vendor. First, the software vendor should present *public seminars* on the use of the software on a regular basis. Also, the vendor should provide timely *technical support* for specific modeling problems encountered by the user. (A toll-free phone number is desirable.) *Good documentation*, including a well-written textbook, a user's manual, and numerous detailed examples, is important for software use as well as initial installation. *Free software trials* and *demo disks* are helpful to the prospective user in evaluating the software for their particular needs.

### 3.4.5 Output Reports

A simulation package should provide time-saving *standard reports* for commonly occurring performance statistics (e.g., utilizations, queue sizes and delays, and throughput), but should also allow *tailored reports* to be developed easily. For example, standard reports are often not suitable for management presentations. Furthermore, it is often of interest to obtain (static) *presentation-quality graphical displays* [e.g., histograms, bar charts, pie charts, or time plots of important variables (see Sec. 9.8)] and to have *access to the individual model output observations* (rather than just the usual summary statistics) so that additional analyses can be performed. For example, one might want to export the output observations (e.g., daily throughputs) to a graphics package, a spreadsheet, or a statistics package.

## 3.5 GPSS

GPSS (General-Purpose Simulation System) is a process-oriented simulation language [see, for example, Gordon (1975) and Schriber (1974)] that is well suited for queueing systems. Originally developed by Geoffrey Gordon at the IBM Corporation in 1961, it evolved through a number of versions, with the most recent IBM version being GPSS V. In the 1960s and 1970s, GPSS was a very popular simulation language, probably due to the queueing nature of

many simulation models, IBM's strong influence on the computer industry, and GPSS's being taught in many university simulation courses. IBM stopped enhancing and actively supporting GPSS in 1972, with the void eventually being filled by the introduction of GPSS/H and GPSS/PC by other vendors. These improved versions of GPSS are described in the following sections.

### 3.5.1 GPSS/H

GPSS/H [see Banks, Carson, and Sy (1989) and Schriber (1990)] was developed by James Henriksen in 1977 and is marketed by Wolverine Software (Annandale, Virginia). GPSS/H is a compiled language, compared with the interpretive approach of GPSS V, and is reported to run, on the average, five times faster [see Abed, Barta, and McRoberts (1985)]. It has a number of other significant enhancements relative to GPSS V, including a real-valued clock, ability to read and write external files, tailored output reports, improved control statements (e.g., DO loops and IF-THEN-ELSE logic), mathematical functions, and a limited number of routines for generating random values from probability distributions. Because of these capabilities and the basic nature of GPSS statements, most GPSS/H models do not require the use of external routines (in FORTRAN or other languages). The random-number generator has also been improved, allowing for an essentially unlimited number of nonoverlapping streams. PROOF [see Brunner and Henriksen (1989)] is a playback-oriented animation package that is marketed by Wolverine Software. It has several interesting features, such as the ability to change quickly from a plan (top) view to an isometric view and the capability to be used with simulation packages developed by several different vendors.

The GPSS/H language consists of more than 60 standard statements, many of which have a corresponding pictorial representation (called a *block*) that is intended to be suggestive of the operation performed by the statement. Building a GPSS model can be thought of as combining a set of standard blocks into a block diagram that represents the path taken by a typical entity as it progresses through the system. After the block-diagram model has been constructed, it is translated by the user into the corresponding set of GPSS statements for execution on the computer. However, the block diagram itself may be useful in explaining the nature of the model to a manager, who may not be familiar with any programming language. Customers or entities that require service of some kind from the system of interest are called *transactions* in GPSS, and their attributes are called *parameters*. The servers or resources that provide the service required by the transactions are called *facilities* or *storages*, corresponding to a single server or a group of parallel servers, respectively.

### 3.5.2 Simulation of the $M/M/1$ Queue

A block diagram and a statement listing for a GPSS/H program of the $M/M/1$ queue (see Sec. 1.4.3) are given in Figs. 3.3 and 3.4, respectively. (The

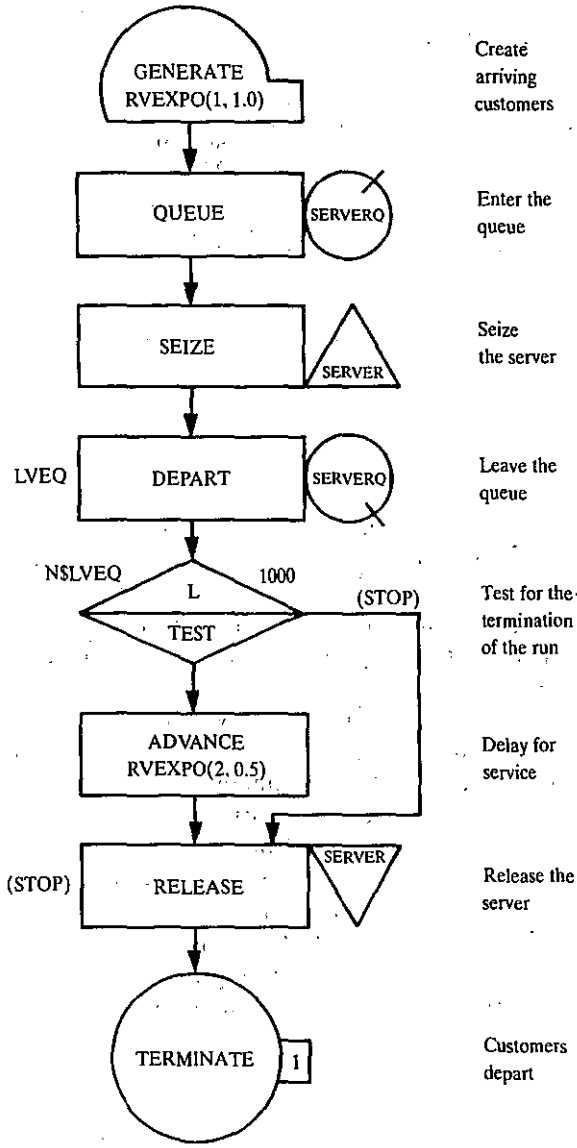



**FIGURE 3.3**
GPSS/H block diagram, queueing model.

program was provided by Professor Thomas Schriber of The University of Michigan.) In Fig. 3.4, statements with an asterisk (*) in column 1 are comments. Also, in lines 5 through 16 the words after position 37 are comments. Line numbers are not part of the program.

The SIMULATE statement (line 4 of the program) is a control statement necessary for program execution. The GENERATE statement (line 5) creates transactions representing customers with exponential (RVEXPO) interarrival times having mean 1.0 and using random-number stream 1. The SEIZE

```
 1  *
 2  *       SIMULATION OF THE M/M/1 QUEUE
 3  *
 4          SIMULATE
 5          GENERATE    RVEXPO(1,1.0)          Create arriving customers
 6          QUEUE       SERVERQ                Enter the queue
 7          SEIZE       SERVER                 Seize the server
 8    LVEQ  DEPART      SERVERQ                Leave the queue
 9          TEST L      N$LVEQ,1000,STOP       Test for termination of the ru
10          ADVANCE     RVEXPO(2,0.5)          Delay for service
11    STOP  RELEASE     SERVER                 Release the server
12          TERMINATE   1                      Customers depart
13  *
14  *       CONTROL STATEMENTS
15  *
16          START       1000                   Make 1 simulation run
17          END
```

**FIGURE 3.4**
GPSS/H program, queueing model.

statement (line 7) and the RELEASE statement (line 11), which define a *facility* called SERVER, correspond to a transaction's seizing the server when it is (or becomes) idle and releasing the server after the transaction's service has been completed. (Transactions arriving when the server is busy join a queue that is automatically defined by GPSS.) The actual service time of a transaction, which is in this case generated from an exponential distribution with mean 0.5 (using stream 2), is experienced at the ADVANCE statement (line 10). The transaction is destroyed (removed from the system) at the TERMINATE statement (line 12). The QUEUE statement (line 6) and the DEPART statement (line 8) are used to gather statistics on transactions waiting in the queue (called SERVERQ) "in front of" facility SERVER, and correspond to a customer's entering and leaving the queue, respectively.

The TEST statement (line 9) is used to determine when to end the simulation run. If the number of transactions, N$LVEQ, that have entered the DEPART block labeled LVEQ (equivalently, have left the queue) is less than 1000, the transaction proceeds to the ADVANCE statement in a normal manner. Otherwise, the transaction is sent to the RELEASE statement labeled STOP, where the server is released without a service time's occurring. Each of the 1000 transactions that enter the TERMINATE statement decrement a counter by 1. Since the termination counter was initially set to 1000 by the START (control) statement (line 16), the 1000th transaction's decrementing the counter reduces the counter value to 0 and results in the termination of the simulation.

The GPSS/H standard output report for this program is given in Fig. 3.5. Note that the average delay is 0.614 (see "AVERAGE TIME/UNIT" for queue SERVERQ). Also, the time-average number in queue (see "AVERAGE CONTENTS" for queue SERVERQ) and server utilization (see "...TOTAL TIME" for facility SERVER) are 0.605 and 0.516, respectively. Server utilization is *automatically* provided when a facility (e.g., SERVER) is

RELATIVE CLOCK: 1014.1565  ABSOLUTE CLOCK: 1014.1565

| BLOCK CURRENT | TOTAL |
|---|---|
| 1 | 1000 |
| 2 | 1000 |
| 3 | 1000 |
| LVEQ | 1000 |
| 5 | 1000 |
| 6 | 999 |
| STOP | 1000 |
| 8 | 1000 |

| FACILITY | --AVG-UTIL-DURING-- TOTAL TIME | AVAIL TIME | UNAVL TIME | ENTRIES | AVERAGE TIME/XACT | CURRENT STATUS | PERCENT AVAIL | SEIZING XACT | PREEMPTING XACT |
|---|---|---|---|---|---|---|---|---|---|
| SERVER | 0.516 | | | 1000 | 0.523 | AVAIL | | | |

| QUEUE | MAXIMUM CONTENTS | AVERAGE CONTENTS | TOTAL ENTRIES | ZERO ENTRIES | PERCENT ZEROS | AVERAGE TIME/UNIT | $AVERAGE TIME/UNIT | QTABLE NUMBER | CURRENT CONTENTS |
|---|---|---|---|---|---|---|---|---|---|
| SERVERQ | 8 | 0.605 | 1000 | 454 | 45.4 | 0.614 | 1.124 | | 0 |

| RANDOM STREAM | ANTITHETIC VARIATES | INITIAL POSITION | CURRENT POSITION | SAMPLE COUNT | CHI-SQUARE UNIFORMITY |
|---|---|---|---|---|---|
| 1 | OFF | 100000 | 101001 | 1001 | 0.71 |
| 2 | OFF | 200000 | 200999 | 999 | 0.69 |

**FIGURE 3.5**
GPSS/H standard output report, queueing model.

defined by the SEIZE and RELEASE statements. The other two statistics result from the use of the QUEUE and DEPART statements.

### 3.5.3 GPSS/PC

GPSS/PC [see Minuteman (1988)] is a simulation language designed specifically for use on the IBM PC and compatibles. It was developed by Springer Cox in 1984 and is marketed by Minuteman Software (Stow, Massachusetts). It has several nice debugging features, including on-line input error checking, on-line help, and the ability to see transactions flowing through the block diagram graphically. Because GPSS/PC is not a compiler, changes made to a model are seen "immediately," without waiting for the program to be recompiled. There are also useful graphical displays for facilities, storages, and histograms, which are updated *dynamically* during the execution of the simulation. GPSS/PC comes standard with concurrent character-graphics animation. An optional three-dimensional, bit-mapped graphics animation capability is also available for use in a playback mode. On the other hand, it has limited facilities for generating random values from probability distributions. One is more likely to need external routines in GPSS/PC than in GPSS/H to perform complex decision logic or produce tailored reports. Also, GPSS/PC is not completely compatible with minicomputer and mainframe versions of GPSS. GPSS/PC has the same *basic* modeling elements (e.g., transactions and facilities) as GPSS/H.

## 3.6 SIMAN/Cinema

SIMAN (SIMulation ANalysis) is a simulation language in which one can build a process-oriented model, an event-oriented model, or a combination of the two [see Pegden, Sadowski, and Shannon (1990)]. In a typical application, most of the simulation model is developed using the process orientation. Complicated decision logic, which is impossible or inconvenient in the process approach, can be coded in event routines and then called from the process model. SIMAN was developed by Dennis Pegden in 1982 and is distributed by Systems Modeling Corporation (Sewickley, Pennsylvania). SIMAN gained quick acceptance because it was the first major simulation language to be available for microcomputers and also because of its special features for manufacturing, including work stations, transporters (e.g., a fork-lift truck), conveyors, and automated guided vehicles. Cinema is a simulation language that contains all of the features of SIMAN and, in addition, the capability to produce high-quality animation. The latest releases of these languages are called SIMAN IV and Cinema IV.

A SIMAN *process* simulation model is broken into two distinct parts, a model frame and an experimental frame, which are kept is separate files. In the *model frame*, modeling constructs called *blocks* are used to describe the logic by which the model's entities and resources interact dynamically. Each block

has a corresponding pictorial representation, and these symbols can be combined into a linear top-down *block diagram*, which graphically describes the flow of entities through the system. Some analysts prefer to construct a block diagram before coding the actual model-frame statements.

In the *experimental frame*, modeling constructs called *elements* are used to specify the particular parameter values (e.g., mean service time) for the present simulation run(s), to define resource types and quantities, and to delineate the output statistics desired. This model/experiment dichotomy may allow the analyst to make two distinct runs of the simulation, perhaps differing only in some parameter value, without recompiling the model frame.

The SIMAN Output Processor allows one to perform certain statistical procedures such as confidence intervals and hypothesis tests on the output data produced by simulation runs from the same or different system configurations. Additionally, it can be used to produce presentation-quality graphical displays such as time plots of variables, histograms, and bar charts. Furthermore, the analyst can choose the desired output data treatments *after* the simulation runs have been made.

SIMAN is available for all major classes of computers. However, with the microcomputer version, it is possible to use an interactive graphical preprocessor called BLOCKS to build the (process-orientation) block diagram. The diagram is then automatically translated into the statement model for execution on the computer. A similar program called ELEMENTS can be used to develop the experimental frame. This capability can increase the speed and accuracy of the model-development process.

The major modeling building blocks in SIMAN are entities (with attributes), queues (or files), and resources.

### 3.6.1 Simulation of the $M/M/1$ Queue

This section shows how to simulate the $M/M/1$ queue considered in Sec. 1.4.3 using the process orientation of SIMAN. A block diagram is given in Fig. 3.6 and the corresponding model-frame statements are given in Fig. 3.7, where the line numbers are for expository purposes and are not part of the program. The CREATE block (line 2) places new customers in the system with exponential [EX(1, 1)] interarrival times, with the first "1" in the parentheses specifying that the mean interarrival time is given by parameter set 1 (see the "1.0" in line 5 of the experimental frame in Fig. 3.8) and the second "1" giving the random-number stream. The modifier MARK(1) stores the time of arrival of a customer in its attribute 1 for later use.

When a customer actually arrives to the system, it temporarily passes through the QUEUE block (line 3) and attempts to seize the resource SERVER (line 4), which is defined in line 4 of the experimental frame. By default, there is one unit of SERVER available. If the server is available, the customer has its zero delay in queue computed and recorded by the TALLY block in line 5 (as the current time minus its time of arrival in attribute 1) and

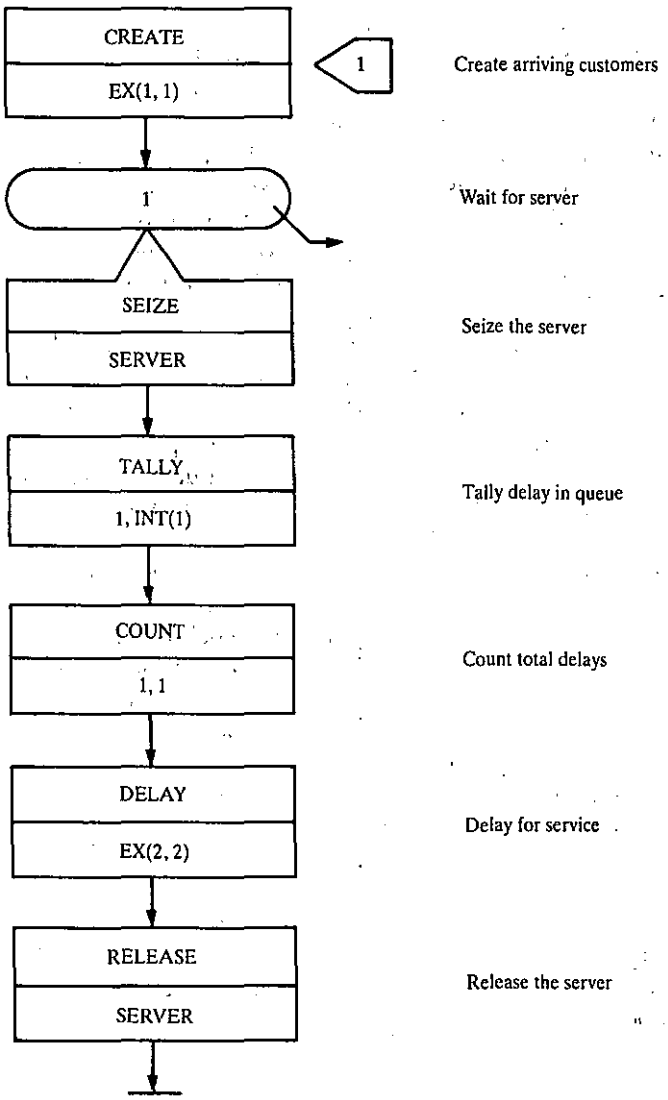



**FIGURE 3.6**
SIMAN block diagram, queueing model.

```
1  BEGIN;
2             CREATE,,EX(1,1):EX(1,1):MARK(1);     Create arriving customers
3             QUEUE,1;                             Wait for server
4             SEIZE:SERVER;                        Seize server
5             TALLY:1,INT(1);                      Tally delay in queue
6             COUNT:1,1;                           Count total delays
7             DELAY:EX(2,2);                       Delay for service
8             RELEASE:SERVER:DISPOSE;              Release server
9  END;
```

**FIGURE 3.7**
SIMAN model frame, queueing model.

```
 1  BEGIN;
 2  PROJECT,M M 1 QUEUE,A. LAW,7/12/89;
 3  DISCRETE,100,1,1;
 4  RESOURCES:1,SERVER;
 5  PARAMETERS:1,1.0:
 6             2,0.5;
 7  TALLIES:1,DELAY IN QUEUE;
 8  COUNTER:1,CUSTOMER DELAYS,1000;
 9  DSTAT:1,NQ(1),NUMBER IN QUEUE:
10        2,NR(1),SERVER UTIL.;
11  REPLICATE,1;
12  END;
```

**FIGURE 3.8**
SIMAN experimental frame, queueing model.

the COUNT block (line 6) adds one to counter 1 to indicate that one more delay has been observed. When this counter reaches 1000 delays, as specified by the COUNTER element in line 8 of Fig. 3.8, the simulation terminates. The customer actually experiences its service time at the DELAY block (line 7); in this case, service times are exponentially distributed with the mean of 0.5 given by parameter set 2 (see line 6 in Fig. 3.8) and are generated using random-number stream 2. [If the server is busy when the above customer arrives, the customer is placed at the end of the queue (file 1) in line 3.] When the customer completes its service, it releases the resource SERVER (line 8) and is removed from the system by the DISPOSE modifier. If there are any customers in the queue, then the first of these is removed, seizes the server (line 4), has its positive delay computed in line 5, etc.

The PROJECT element (line 2) of the experimental frame states the project name, the analyst, and the date. The DISCRETE element (line 3) specifies the computer storage requirements for the model, here being 100 entities (customers) simultaneously in the model, a maximum of 1 attribute for any entity, and 1 queue (file) for the model. The elements in lines 4 through 6 have been explained above. The TALLIES element (line 7) places a label of "DELAY IN QUEUE" on the discrete-time statistics produced by the TALLY block in line 5 of the model frame. The DSTAT element (lines 9 and 10) computes continuous-time statistics (e.g., the time average and the maximum) for the number in queue 1 [NQ(1)] and for the number of busy units of resource 1, NR(1); these functions are referred to as DSTAT variables 1 and 2, respectively. Thus, for example, the time average of variable 2 will be the server utilization. The REPLICATE element (line 11) specifies that one replication of the simulation is to be made. A more general form of this statement can be used to specify multiple replications, a simulation run length, and a warmup period.

The simulation results are given in Fig. 3.9. Note that the average delay in queue is 0.49558 (see "Tally Variables"). Also, the time-average number in queue and server utilization (computed by the DSTAT element) are 0.50658 and 0.51872, respectively (see "Discrete Change Variables"). The "Standard Deviation" results in the output report are not reliable, in general, since they are based on formulas that assume independent output data, which will not be satisfied in practice (see Sec. 4.4).

```
                              SIMAN Summary Report

                              Run Number   1 of  1

Project:  M M 1 QUEUE
Analyst:  A. LAW
Date   :   7/12/1989

Run ended at time     .9783E+03


                                 Tally Variables
                                 ---------------

Number Identifier          Average     Standard    Minimum     Maximum     Number
                                       Deviation   Value       Value       of Obs.
-------------------------------------------------------------------------------

    1  DELAY IN QUEUE         .49558      .80138      .00000     4.04199       1000


                            Discrete Change Variables
                            -------------------------

Number Identifier          Average     Standard    Minimum     Maximum     Time
                                       Deviation   Value       Value       Period
-------------------------------------------------------------------------------

    1  NUMBER IN QUEUE        .50658     1.14931      .00000    10.00000     978.28
    2  SERVER UTIL.           .51872      .49965      .00000     1.00000     978.28


                                   Counters
                                   --------

Number Identifier          Count       Limit
-----------------------------------------
    1  CUSTOMER DELAYS       1000        1000
```

**FIGURE 3.9**
SIMAN output report, queueing model.

Observe that the output statistics for GPSS/H and SIMAN are somewhat different due to differences in the random-number generators used (see also Secs. 3.7 and 3.8). This points out the importance of proper design and analysis of simulation runs, as discussed in Chap. 9.

## 3.7 SIMSCRIPT II.5

SIMSCRIPT II.5 is a process-oriented or event-oriented simulation language [see Law and Larmey (1984) and Russell (1983)]; however, because of the generality of the process approach in SIMSCRIPT, the use of the event-scheduling approach is not necessary. SIMSCRIPT was developed by Harry Markowitz and others at the Rand Corporation in 1962. It evolved through a number of versions, with the latest one, SIMSCRIPT II.5, being marketed by CACI Products Company (La Jolla, California).

SIMSCRIPT II.5 is actually a general programming language containing the capabilities for building discrete-event, continuous, or combined simulation models. (It has the programming features of FORTRAN, ALGOL, and PL/I.) Furthermore, its English-like and free-form syntax make SIMSCRIPT II.5 simulation programs easy to read and almost self-documenting. Because of its general process approach, its sophisticated data structures, and its powerful control statements, SIMSCRIPT II.5 is often used for large, complex simulation models, particularly when the system is not queueing-oriented. For example, most military combat models have been written either in SIMSCRIPT II.5 or FORTRAN.

SIMSCRIPT II.5 is available for microcomputers, work stations, and minicomputers/mainframes. The IBM PC and compatibles version is embedded in the SIMLAB package, which is an interactive, multitasking programming environment for facilitating the use of SIMSCRIPT. It contains an editor, the SIMSCRIPT II.5 compiler, a debugger, and on-line help.

The microcomputer and work station versions include the SIMGRAPHICS animation and graphics package. It can be used to produce both dynamic and static presentation-quality graphics, such as histograms, pie and bar charts, level meters and dials, and time plots of variables. Animations of the simulation output are also constructed using SIMGRAPHICS. Finally, SIMGRAPHICS can be used to produce interactive graphical "front ends" (or forms) for entering model input data. An input form may include such graphical elements as menubars (with pull-down menus), text or data "boxes," and "buttons" that are clicked on with a mouse to select an alternative. The graphical model front end allows one to make a certain set of modifications to the model without programming, which facilitates model use by people who are not programming experts.

The major modeling elements of the process part of SIMSCRIPT II.5 are processes (or process entities), resources, and sets (similar to queues). A process entity flows through its corresponding process and may have attributes. To construct a simulation model in SIMSCRIPT II.5, the analyst must write a preamble, a main program, and a process routine corresponding to each process. The *preamble*, which does not contain any executable statements, is used to define the building blocks for the simulation, such as processes and resources. It is also used to define global variables, the basic unit of time for the simulation clock, and the desired output statistics. In the latter case, the TALLY and ACCUMULATE statements are used to specify discrete-time and continuous-time statistics, respectively. The main program is where the execution of a SIMSCRIPT program begins. This routine is used to read input parameters for the simulation, to specify the number of available units for each resource, and to place the "initial" event records (called *process notices*) into the event list using the ACTIVATE statement. The simulation actually begins by executing the START SIMULATION statement, which is actually just a call to the timing routine. The timing routine is part of the SIMSCRIPT II.5 language and does not have to be written by the modeler.

### 3.7.1 Simulation of the $M/M/1$ Queue

This section shows how to simulate the $M/M/1$ queue of Sec. 1.4.3 using the process orientation of SIMSCRIPT II.5. (The line numbers in Figs. 3.10 through 3.14 are for expository purposes and are not part of the actual program.) The preamble is given in Fig. 3.10. Three types of processes, ARRIVAL.GENERATOR, CUSTOMER, and REPORT, are defined in line 3. Process (routine) CUSTOMER describes the flow of a typical customer as it moves through the system. On the other hand, process (routine) ARRIVAL.GENERATOR creates new customers, and process (routine) REPORT is used to print the final report at the end of the simulation after 1000 delays in queue have been completed. Similarly, SERVER is defined to be a resource in line 5, and has the two associated sets, Q.SERVER (customers in queue for SERVER) and X.SERVER (customers executing on SERVER), automatically specified. The three quantities in lines 7 and 8 are defined to be *global* real variables. (If a variable is not defined in the preamble, it is a local variable. Also, all variables are by default real, regardless of the letter they begin with.) The desired simulation run length in delays, TOT.DELAYS, is defined to be a global integer variable in line 10. In line 12, MINUTES is defined (in effect) to be the basic unit of time for internal program calculations; the default is days. The TALLY statement (lines 14 and 15) is used to obtain discrete-time statistics for the variable DELAY.IN.QUEUE. In particular, NUM.DELAYS will be the number of delays observed (i.e., the number of times that a statement with DELAY.IN.QUEUE on the left-hand side of an equal sign is executed), and AVG.DELAY.IN.QUEUE will be the sample mean of these delays. The ACCUMULATE statement (line 17) is used to compute continuous-time statistics on the system-defined variable N.Q.SER-

```
 1  PREAMBLE
 2
 3     PROCESSES INCLUDE ARRIVAL.GENERATOR, CUSTOMER, AND REPORT
 4
 5     RESOURCES INCLUDE SERVER
 6
 7     DEFINE DELAY.IN.QUEUE, MEAN.INTERARRIVAL.TIME, AND
 8        MEAN.SERVICE.TIME AS REAL VARIABLES
 9
10     DEFINE TOT.DELAYS AS AN INTEGER VARIABLE
11
12     DEFINE MINUTES TO MEAN UNITS
13
14     TALLY AVG.DELAY.IN.QUEUE AS THE AVERAGE AND NUM.DELAYS AS
15        THE NUMBER OF DELAY.IN.QUEUE
16
17     ACCUMULATE AVG.NUMBER.IN.QUEUE AS THE AVERAGE OF N.Q.SERVER
18
19     ACCUMULATE UTIL.SERVER AS THE AVERAGE OF N.X.SERVER
20
21  END
```

**FIGURE 3.10**
SIMSCRIPT II.5 preamble, queueing model.

VER, which is the number of customers in the set Q.SERVER at a particular point in time. The quantity AVG.NUMBER.IN.QUEUE will be the time average of N.Q.SERVER over the length of the simulation. The system-defined variable N.X.SERVER in line 19 is the number of customers in the set X.SERVER at a particular point in time, which can be 1 or 0 in our case. Thus, if we use the ACCUMULATE statement to compute the time average of this variable over the length of the simulation, we obtain the proportion of time UTIL.SERVER that the server is busy.

The main program is listed in Fig. 3.11. In line 3, a free-format READ statement is used to read in the input parameters MEAN.INTERARRIVAL.TIME (=1.0), MEAN.SERVICE.TIME (=0.5), and TOT.DELAYS (=1000). The CREATE statement (line 5) specifies that there is one type of the resource SERVER. (Each type of a resource is fed by a single queue.) In line 6 the number of available units of the first (and in this case only) type of resource SERVER, namely, U.SERVER(1), is set to 1. The ACTIVATE statement (line 8) places an ARRIVAL.GENERATOR process notice into the event list with an event time (called an *activation time*) of "NOW." Time "NOW" means that the process notice has an activation time equal to the current value of simulated time, TIME.V (=0 in this instance), and that this process notice is placed "first" in the event list. This process notice is used to initialize the ARRIVAL.GENERATOR process routine at time 0. The START SIMULATION statement (line 10) calls the timing routine and begins the execution of the simulation. The timing routine will remove the first process notice from the event list, which in this case will be the one corresponding to the ARRIVAL.GENERATOR process.

The ARRIVAL.GENERATOR process routine is listed in Fig. 3.12. [We will not explain its exact operation here; see Law and Larmey (1984, pp. 2–12) for details.] It is used to cause new customers to arrive to the system with exponential interarrival times having mean MEAN.INTERARRIVAL.TIME minutes using random-number stream 1 (line 5). At the time instant that a particular customer is to arrive, the ARRIVAL.GENERATOR routine places a CUSTOMER process notice in the event list with an activation time of NOW (line 6). This causes the timing routine to call the CUSTOMER process

```
 1  MAIN
 2
 3      READ MEAN.INTERARRIVAL.TIME, MEAN.SERVICE.TIME, AND TOT.DELAYS
 4
 5      CREATE EVERY SERVER(1)
 6      LET U.SERVER(1) = 1
 7
 8      ACTIVATE AN ARRIVAL.GENERATOR NOW
 9
10      START SIMULATION
11
12  END
```

**FIGURE 3.11**
SIMSCRIPT II.5 main program, queueing model.

```
1  PROCESS ARRIVAL.GENERATOR
2
3     WHILE TIME.V >= 0.0
4     DO
5        WAIT EXPONENTIAL.F(MEAN.INTERARRIVAL.TIME,1) MINUTES
6        ACTIVATE A CUSTOMER NOW
7     LOOP
8
9  END
```

**FIGURE 3.12**
SIMSCRIPT II.5 process routine ARRIVAL.GENERATOR, queueing model.

routine immediately in order to process the newly arriving customer. The SIMSCRIPT II.5 ARRIVAL.GENERATOR routine corresponds to the CREATE statement in SIMAN and SLAM II and to the GENERATE statement in GPSS.

The process routine for process CUSTOMER is given in Fig. 3.13, and is called each time a process notice for a customer process entity is removed from the event list, corresponding to the arrival of a new customer. [This routine is also called in several other situations, such as when a customer departs; see Law and Larmey (1984) for details.] The DEFINE statement in line 3 specifies that TIME.OF.ARRIVAL is a local real variable (associated with each customer). The time of arrival of the currently arriving customer is set to the current value of the simulation clock, TIME.V, in line 5. The customer then requests one unit of the resource SERVER(1) in line 6. If the server is already busy serving another customer entity, the arriving customer joins the queue Q.SERVER(1) and waits to be served at some point in the future. If the server is idle, the delay in queue of the arriving customer is set to 0 in line 7 and a check for termination of the simulation run is made in lines 8 through 10 (to be discussed below). The server then *works* on the service request of the customer (line 11), whose service duration is generated from an exponential distribution with mean MEAN.SERVICE.TIME minutes using stream 2. After this customer's service has been completed, the customer *relinquishes* the server in line

```
 1  PROCESS CUSTOMER
 2
 3     DEFINE TIME.OF.ARRIVAL AS A REAL VARIABLE
 4
 5     LET TIME.OF.ARRIVAL = TIME.V
 6     REQUEST 1 SERVER(1)
 7     LET DELAY.IN.QUEUE = TIME.V - TIME.OF.ARRIVAL
 8     IF NUM.DELAYS = TOT.DELAYS
 9        ACTIVATE A REPORT NOW
10     ALWAYS
11     WORK EXPONENTIAL.F(MEAN.SERVICE.TIME,2) MINUTES
12     RELINQUISH 1 SERVER(1)
13
14  END
```

**FIGURE 3.13**
SIMSCRIPT II.5 process routine CUSTOMER, queueing model.

12 and is then removed from the system. If any customers are in Q.SERVER(1) when the server becomes available, the first customer is removed and experiences a positive delay in queue in line 7, etc.

Lines 8 through 10 of process routine CUSTOMER are used to determine when to terminate the simulation run. If NUM.DELAYS (defined in the preamble) is equal to TOT.DELAYS (=1000), then a REPORT process notice is placed in the event list with an activation time of NOW. Control is then returned to the timing routine, which immediately calls the REPORT process routine to terminate the simulation run.

The REPORT process routine is listed in Fig. 3.14, and is called by the timing routine when 1000 customer delays have been completed. The PRINT statement (line 3), which contains no variables, specifies that the five lines following this statement (the first three of which are blank) are printed out exactly as shown. The PRINT statement in lines 9 and 10 says that the three specified variables will be printed out in eight lines exactly as shown. The formats for the three variables are given by the three successive asterisk groups. Thus, the format for MEAN.INTERARRIVAL.TIME is "**.**," which means that the corresponding printed value will be real-valued, have two places to the right of the decimal point, etc. The PRINT statement in lines 19 and 20 is similar. Note, however, that the resource type is specified explicitly in

```
    1  PROCESS REPORT
    2
    3      PRINT 5 LINES THUS


SIMULATION OF THE M/M/1 QUEUE

    9      PRINT 8 LINES WITH MEAN.INTERARRIVAL.TIME, MEAN.SERVICE.TIME,
   10          AND TOT.DELAYS THUS

MEAN INTERARRIVAL TIME      **.**

MEAN SERVICE TIME           **.**

NUMBER OF CUSTOMERS         *****

   19      PRINT 8 LINES WITH AVG.DELAY.IN.QUEUE, AVG.NUMBER.IN.QUEUE(1),
   20          AND UTIL.SERVER(1) THUS

AVERAGE DELAY IN QUEUE      ***.**

AVERAGE NUMBER IN QUEUE     ***.**

SERVER UTILIZATION            *.**

   29      STOP
   30
   31  END
```

**FIGURE 3.14**
SIMSCRIPT II.5 process routine REPORT, queueing model.

SIMULATION OF THE M/M/1 QUEUE

| | |
|---|---|
| MEAN INTERARRIVAL TIME | 1.00 |
| MEAN SERVICE TIME | .50 |
| NUMBER OF CUSTOMERS | 1000 |
| AVERAGE DELAY IN QUEUE | .43 |
| AVERAGE NUMBER IN QUEUE | .43 |
| SERVER UTILIZATION | .50 |

**FIGURE 3.15**
SIMSCRIPT II.5 output report, queueing model.

the output. For example, AVG.NUMBER.IN.QUEUE(1) is the time average of Q.SERVER(1). Finally, execution of the STOP statement (line 29) will terminate the simulation, as desired.

The SIMSCRIPT II.5 output report, as printed by process routine REPORT, is given in Fig. 3.15.

## 3.8 SLAM II AND RELATED SOFTWARE

SLAM II (Simulation Language for Alternative Modeling) is a simulation language in which one can build a process-oriented model, an event-oriented model, or a combination of the two [see Pritsker (1986)]. In a typical application, most of the simulation model is developed using the process orientation. Complicated decision logic, which is impossible or inconvenient in the process approach, is coded in event routines and then called from the process model. SLAM was developed by Dennis Pegden and Alan Pritsker in 1979 and is distributed by the Pritsker Corporation (Indianapolis, Indiana).

The building of a process model often begins with the analyst developing a graphical network diagram for the system. This diagram is constructed by combining a standard set of symbols, called *nodes* and *branches*, into an interconnected *network* that represents the flow of an entity through its corresponding process. A node may correspond, for example, to the creation of entities or to a queue, while a branch may correspond to the passage of time (e.g., a service time). The network model of the system is then translated into an equivalent set of SLAM II program statements for execution on the computer. The program statements could also be coded directly, without a network diagram.

SLAM II is available in several different forms, depending on the computer platform and whether an animation capability is desired. The basic SLAM II language is available for all computer classes, but does not include animation. SLAMSYSTEM is a microcomputer version of SLAM II that is integrated with Microsoft Windows. It provides animation, presentation-qual-

ity graphics (e.g., time plots of variables, histograms, bar charts, and pie charts), and a user-friendly environment. SLAM II/TESS is available for engineering work stations, minicomputers, and mainframes. It has animation and graphics capabilities similar to SLAMSYSTEM and, in addition, contains an integrated database for model input/output and enhanced statistical features such as confidence intervals.

With SLAMSYSTEM or SLAM II/TESS, one can graphically build the SLAM II network diagram on a CRT, which is then automatically translated into the corresponding program statements for execution by SLAM II. This feature can increase the speed and accuracy of the modeling process.

There is a Material Handling Extension to SLAM II that allows one to simulate automated guided vehicle systems, cranes, and automated storage and retrieval systems.

A SLAM II process simulation model is coded in a single integrated subprogram. Discrete-time statistics (e.g., average and maximum delay) are obtained in SLAM II using the COLCT node. On the other hand, continuous-time statistics on queues (e.g., average length) and resources (e.g., utilization) are provided *automatically*. The major modeling elements in SLAM II are entities (with attributes), files (or queues), and resources.

### 3.8.1 Simulation of the $M/M/1$ Queue

This section presents a SLAM process model for the $M/M/1$ queue of Sec. 1.4.3. The network diagram and statement model are given in Figs. 3.16 and 3.17, respectively; the line numbers in Fig. 3.17 are for expository purposes and are not part of the program. The GEN (general) control statement in line 1 states the analyst, the project name, the date, the number of runs (i.e., 1), and the number of columns for output reports (i.e., 72), respectively. (The successive commas represent accepted defaults.) The LIMITS control statement (line 2) declares that the model will contain 1 file (queue), a maximum of 1 attribute per entity, and that no more than 100 entities will be present in the model simultaneously. The NETWORK and END statements in lines 4 and 17 signify the start and end of the process (network) model.

The RESOURCE block in line 6 defines a resource named SERVER with a capacity of 1 unit as specified in the parentheses. The second "1" states that when the SERVER is available, it will serve the first customer in file 1 (see line 9) next. (Lines beginning with semicolons are comments.) The CREATE node (line 8) places new customers in the system with interarrival times $2, 3, \ldots$ being exponentially (EXPON) distributed with a mean of 1.0 and using random-number stream 1. The first 1 after the right parenthesis states that the first interarrival time is exactly 1. (The CREATE node in SLAM II does not allow the first interarrival time to be a random variable; this difficulty could be overcome by adding two additional lines of code.) The next 1 places the time of arrival of each arriving entity in its attribute 1. The AWAIT node (line 9) corresponds to the resource SERVER and its preceding queue. If a

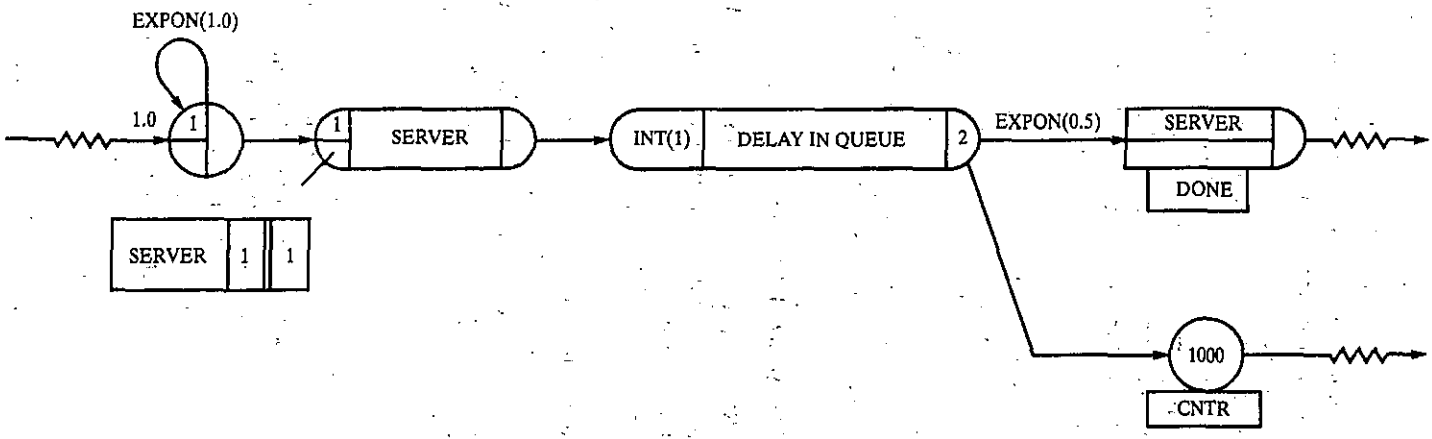



**FIGURE 3.16**
SLAM II network diagram, queueing model.

```
1  GEN,A. LAW,M M 1 QUEUE,7/13/1989,1,,,,,,72;
2  LIM,1,1,100;
3  ;
4  NETWORK;
4  ;
6            RESOURCE/SERVER(1),1;              Define the resource server
7  ;
8            CREATE,EXPON(1.0,1),1,1;           Create arriving customers
9            AWAIT(1),SERVER;                   Wait for/seize server
10           COLCT,INT(1),DELAY IN QUEUE,,2;    Collect delay in queue
11           ACTIVITY,EXPON(0.5,2),,DONE;       Delay for service
12           ACTIVITY,,,CNTR;                   Send "dummy" entity to counter
13 DONE      FREE,SERVER;                       Release server
14           TERM;                              Customers depart
15 ;
16 CNTR      TERM,1000;                         End simulation after 1000 delays
17           END;
18 ;
19 INIT;
20 FIN;
```

**FIGURE 3.17**
SLAM II program, queueing model.

customer arrives and the SERVER is available, the customer seizes the server immediately and moves to the next line of the program. Otherwise, the customer is placed last in the FIFO queue (file 1). The COLCT node (line 10) computes and records the delay in queue of each customer as the current time minus its time of arrival in attribute 1. Strictly speaking, it is not needed here to compute the average delay in queue, since this will be done automatically by the AWAIT node. However, the COLCT node is needed in general to obtain discrete-time statistics (e.g., maximum delay in queue) and is included here to illustrate its use. The 2 at the end of this line creates a duplicate copy of the entity, with one entity's being routed to line 11 and the other to line 12 (to be discussed after line 14). The entity arriving to line 11 corresponds to the actual customer moving through the system. The ACTIVITY branch there is where the customer actually experiences its service time, which is generated from an exponential distribution with mean 0.5 using stream 2. When the customer's service is completed, the entity is routed to the statement labeled DONE (line 13). At this line, the FREE node causes the entity to release the server and to move on to line 14 for removal from the system (termination). If there are any customers in the queue (file 1), then the first of these is removed and seizes the server in line 9, etc.

The "dummy" entity arriving to line 12 is used to terminate the simulation in the appropriate manner. It is immediately sent to the TERMINATE node in line 16 (labeled CNTR), which adds one to a counter to indicate that one more delay has been observed. When this counter reaches 1000 delays, the simulation is terminated.

The simulation results are given in Fig. 3.18. Note that the average delay in queue is 0.608 (see "STATISTICS FOR VARIABLES BASED ON OBSERVATION"), as computed by the COLCT node. (See also "AVERAGE

```
                    S L A M   I I   S U M M A R Y   R E P O R T

SIMULATION PROJECT M M 1 QUEUE                    BY A. LAW

DATE  7/13/1989                                   RUN NUMBER    1 OF    1

CURRENT TIME     .9171E+03
STATISTICAL ARRAYS CLEARED AT TIME    .0000E+00


            **STATISTICS FOR VARIABLES BASED ON OBSERVATION**

                     MEAN      STANDARD  COEFF. OF  MINIMUM    MAXIMUM   NO.OF
                     VALUE     DEVIATION VARIATION  VALUE      VALUE     OBS

DELAY IN QUEUE     .608E+00   .100E+01  .165E+01   .000E+00   .589E+01  1000


            **FILE STATISTICS**

FILE                        AVERAGE   STANDARD   MAXIMUM  CURRENT AVERAGE
NUMBER  LABEL/TYPE          LENGTH    DEVIATION  LENGTH   LENGTH  WAIT TIME

   1          AWAIT           .662      1.354       9        0       .608
   2          CALENDAR       1.555       .497       3        2       .285


            **RESOURCE STATISTICS**

RESOURCE  RESOURCE  CURRENT   AVERAGE   STANDARD   MAXIMUM  CURRENT
NUMBER    LABEL     CAPACITY  UTIL      DEVIATION  UTIL     UTIL

    1     SERVER          1       .55        .497       1        1


RESOURCE  RESOURCE  CURRENT    AVERAGE      MINIMUM     MAXIMUM
NUMBER    LABEL     AVAILABLE  AVAILABLE    AVAILABLE   AVAILABLE

    1     SERVER          0       .4452           0           1
```

**FIGURE 3.18**
SLAM II output report, queueing model.

WAIT TIME" for file number 1.) In addition, the time-average number in queue (see "AVERAGE LENGTH" for file number 1) and server utilization (see "AVERAGE UTIL" for resource number 1) are 0.662 and 0.55, respectively. These statistics are automatically computed and written out when the AWAIT node is used. The "STANDARD DEVIATION" results in the output report are not reliable, in general, since they are based on formulas that assume independent output data, which will not be satisfied in practice (see Sec. 4.4).

## 3.9 COMPARISON OF SIMULATION LANGUAGES

In this section we briefly discuss and compare the simulation languages presented in Secs. 3.5 through 3.8. These languages actually have very similar basic modeling constructs, due to language cross-fertilization over the years. This can be seen in Table 3.1, where we show the GPSS (H or PC), SIMAN/Cinema, SIMSCRIPT II.5, and SLAM II/SLAMSYSTEM language statements for creating new entities, for entities to seize and release resources, for a passage of time (e.g., a service time), and for collecting discrete-time and continuous-time statistics.

Many simulations have a queueing orientation, and GPSS and the process parts of SIMAN and SLAM II have modeling constructs well suited for these types of problems. SIMAN and SLAM II also have constructs for the more basic event-scheduling approach. This should allow them to model *conveniently* a somewhat larger class of non-queueing-oriented systems than GPSS. On the other hand, there is some indication that GPSS/H has the fastest compilation and execution times [see Abed, Barta, and McRoberts (1985)].

SIMSCRIPT II.5 has the most general process approach of the major simulation languages; thus, virtually any system can be modeled without using the event-scheduling approach. However, because of its general structure, it

**TABLE 3.1**
**Implementation of basic simulation capabilities**

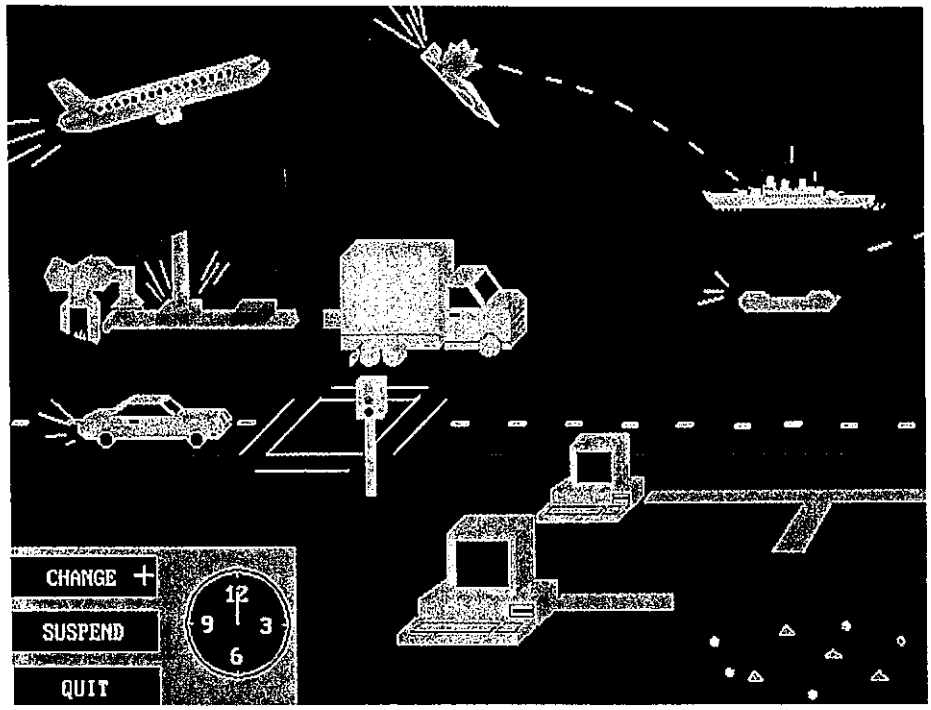
| | Language | | | |
|---|---|---|---|---|
| **Feature** | **GPSS (H or PC)** | **SIMAN/ Cinema** | **SIMSCRIPT II.5** | **SLAM II/ SLAMSYSTEM** |
| Create new entities | GENERATE | CREATE | ACTIVATE | CREATE |
| Seize and release a resource | SEIZE/ RELEASE | SEIZE/ RELEASE | REQUEST/ RELINQUISH | AWAIT/ FREE |
| Passage of time (e.g., a service time) | ADVANCE | DELAY | WORK, WAIT | ACTIVITY |
| Discrete-time statistics | QUEUE/ DEPART, TABULATE | TALLY | TALLY | COLCT[a] |
| Continuous-time statistics | QUEUE/ DEPART, ENTER/ LEAVE, TABULATE[a] | DSTAT | ACCUMULATE | TIMST[a] |

[a] Some statistics provided automatically.

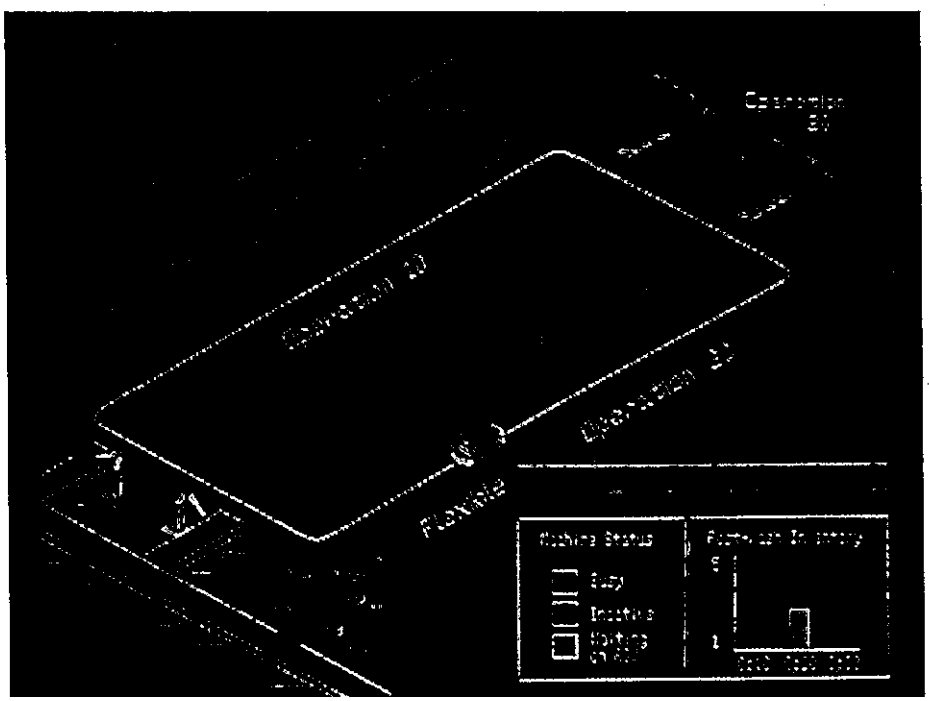
**TABLE 3.2**
**Comparison of the simulation languages**

| Feature | Language | | | | | |
|---|---|---|---|---|---|---|
| | GPSS/H | GPSS/PC | SIMAN/ Cinema | SIMSCRIPT II.5 | SLAM II | SLAMSYSTEM |
| Event (E) or process (P) orientation | P | P | E, P | E, P | E, P | E, P |
| Available for which computer classes? | MICRO,[a] WORK,[b] MIN/MAIN[c] | MICRO | MICRO, WORK, MIN/MAIN | MICRO, WORK, MIN/MAIN | MICRO, WORK, MIN/MAIN | MICRO |
| Animation for which computer classes? | MICRO | MICRO | (MICRO, WORK)[d] | MICRO, WORK | (WORK, MIN/MAIN)[e] | MICRO |
| Graphical model input | No | No | Yes[f] | No | Yes[e] | Yes |
| Combined discrete-continuous simulation | No | Yes | Yes | Yes | Yes | Yes |
| Number of random-number streams | Essentially unlimited | Essentially unlimited | 10[g] | 10[g] | 10[g] | 10 |
| Standard probability distributions[h] | Ex, N, T, U | U | Be, Er, Ex, Ga, L, N, P, T, U, W | Be, Bi, Er, Ex, Ga, L, N, P, T, U, W | Be, Er, Ex, Ga, L, N, P, T, U, W | Be, Er, Ex, Ga, L, N, P, T, U, W |
| Single command for automatic multiple replications | No | No | Yes | No | Yes | Yes |
| Confidence-interval procedures[i] | None | R, BM | R, BM, STS | None | (R, BM)[e] | None |

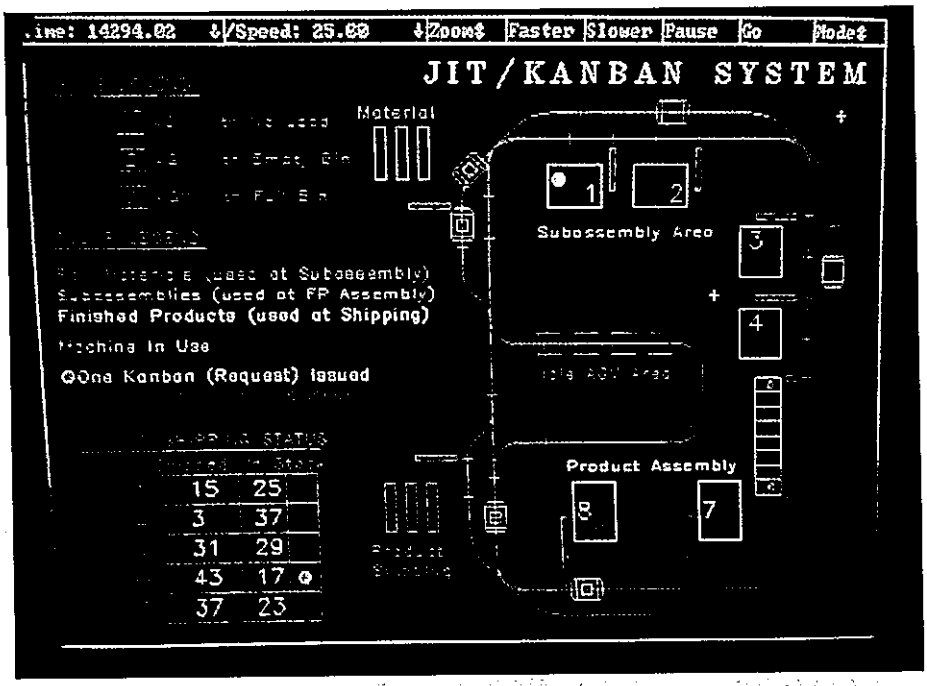
[a] Microcomputer.
[b] Work station.
[c] Minicomputer/mainframe.
[d] Cinema only.
[e] Using SLAM II/TESS.
[f] Microcomputer only.
[g] Extendable.



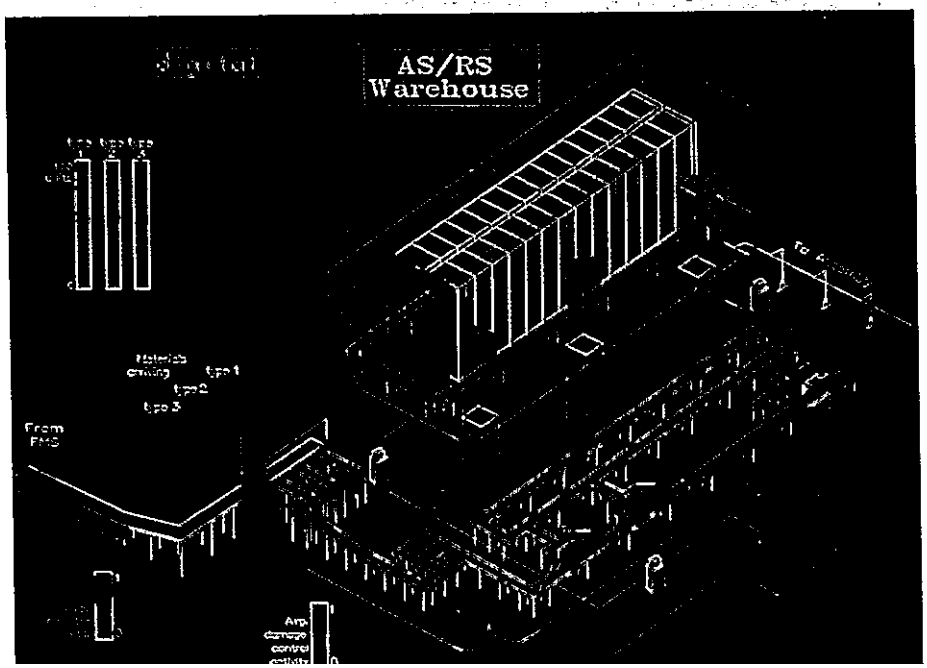

(c)



(d)


Time: 14294.02
Speed: 25.00
Zoom$
Faster
Slower
Pause
Go
Mode$
JIT/KANBAN SYSTEM
Material
Subassembly Area
Finished Products (used at Shipping)
Machine In Use
One Kanban (Request) Issued
Product Assembly
15 25
3 37
31 29
43 17
37 23

(a)


AS/RS
Warehouse
From FMS

(b)

may require more lines of code than GPSS, SIMAN, or SLAM II for "standard" queueing problems. For "complicated" simulation models (particularly those that are large or non-queueing-oriented), SIMSCRIPT II.5 is an attractive choice because it is a general programming language with sophisticated control statements and data structures.

In Sec. 3.4 we discussed a number of features to consider when selecting simulation software. There are two levels at which such a decision could be made. At the first level, an organization must decide what languages or simulators to purchase (or lease) for its general use. The reader should be aware that there is no simulation package that is convenient and appropriate for *all* applications. Thus, organizations that do a large amount of simulation may want to consider having several simulation packages, to be used for different types of applications and by people with different backgrounds. At the second level, an analyst must decide what simulation software to use for a particular study.

Many important simulation software features are quite subjective in nature (e.g., ease of model development and vendor technical support) and, thus, will not be used to compare the simulation languages discussed above. As an alternative, we present in Table 3.2 a comparison of the simulation languages based on nine quantitative features or factors. This list is *not exhaustive*, and whether a feature is important could depend on the particular application. For example, most simulation studies do not require capabilities for combined discrete-continuous simulation. In Table 3.2 a simulation language is said to have a particular feature if it is part of the software usually distributed by the vendor.

Note in Table 3.2 that SIMSCRIPT II.5 does not explicitly provide automatic multiple replications and confidence intervals. However, this capability (including replication and batch-means confidence intervals) is available from CACI in free, optional software [see Law (1979)]. Also, multiple replications can easily be obtained in GPSS/H using a DO loop.

Some additional information on the above simulation languages is given in Banks and Carson (1985). In particular, they provide GPSS/H, SIMAN, SIMSCRIPT II.5, and SLAM II programs for a simple manufacturing system.

## 3.10 ADDITIONAL SIMULATION SOFTWARE

In addition to the simulation languages discussed in the previous sections, there are several others of note, namely, INSIGHT [SysTech (1985)], PCModel [White (1988)], and SIMPLE_1 [Sierra (1989)]. In addition, MODSIM II [Belanger et al. (1989)] and SIM++ [Jade (1989)] are recently introduced simulation languages based on object-oriented programming that promote greater simulation software reusability and will also run on parallel processors.

A large number of simulation packages have been developed specifically for manufacturing applications, including AutoMod II, ProModel, SIMFAC-

TORY II.5, WITNESS, and XCELL+; these products are described in Sec. 13.3.

NETWORK II.5 is a simulator for computer systems and local-area networks [see Cheung, Dimitriadis, and Karplus (1987) and CACI (1988a)]. Its basic building blocks are processing elements (e.g., a CPU), transfer devices (e.g., a bus), storage devices (e.g., a disk drive), and software modules. COMNET II.5, on the other hand, is a simulator for wide-area telecommunication networks [CACI (1988b)]. Its building blocks are the network topology (nodes and their connecting links), network traffic (source, destination, and size of messages), and network operations (strategies for choosing message routes). Both simulators are distributed by CACI Products Company.

## REFERENCES


Abed, S. Y., T. A. Barta, and K. L. McRoberts: A Quantitative Comparison of Three Simulation Languages: GPSS/H, SLAM, SIMSCRIPT, *Comput. Ind. Eng.*, *9*: 45–66 (1985).

Banks, J., and J. S. Carson: Process-Interaction Simulation Languages, *Simulation*, *44*: 225–235 (1985).

Banks, J., J. S. Carson, and J. N. Sy: *Getting Started with GPSS/H*, Wolverine Software Corporation, Annandale, Va. (1989).

Belanger, R. F., B. Donovan, K. L. Morse, S. V. Rice, and D. B. Rockower: *MODSIM II Reference Manual*, CACI Products Company, La Jolla, Calif. (1989).

Brunner, D. T., and J. O. Henriksen: A General Purpose Animator, *Proc. 1989 Winter Simulation Conference*, Washington, D.C., pp. 155–163 (1989).

CACI Products Company: *NETWORK II.5 User's Manual*, La Jolla, Calif. (1988a).

CACI Products Company: *COMNET II.5 User's Manual*, La Jolla, Calif. (1988b).

Cheung, S., S. Dimitriadis, and W. J. Karplus: *Introduction to Simulation Using NETWORK II.5*, CACI Products Company, La Jolla, Calif. (1987).

Gordon, G.: *The Application of GPSS V to Discrete System Simulation*, Prentice-Hall, Englewood Cliffs, N.J. (1975).

Jade Simulations International Corporation: *SIM++ Release 2.0*, Calgary, Alberta (1989).

Law, A. M.: *Statistical Analysis of Simulation Output Data with SIMSCRIPT II.5*, CACI Products Company, La Jolla, Calif. (1979).

Law, A. M., and S. W. Haider: Selecting Simulation Software for Manufacturing Applications: Practical Guidelines & Software Survey, *Ind. Eng.*, *31*: 33–46 (May 1989).

Law, A. M., and C. S. Larmey: *Introduction to Simulation Using SIMSCRIPT II.5*, CACI Products Company, La Jolla, Calif. (1984).

Minuteman Software: *GPSS/PC Reference Manual*, Stow, Mass. (1988).

Pegden, C. D., R. P. Sadowski, and R. E. Shannon: *Introduction to Simulation Using SIMAN*, Systems Modeling Corporation, Sewickley, Pa. (1990).

Pritsker, A. A. B.: *Introduction to Simulation and SLAM II*, 3d ed., Halsted, New York (1986).

Russell, E. C.: *Building Simulation Models with SIMSCRIPT II.5*, CACI Products Company, La Jolla, Calif. (1983).

Schriber, T. J.: *Simulation Using GPSS*, John Wiley, New York (1974).

Schriber, T. J.: *An Introduction to Simulation Using GPSS/H*, John Wiley, New York (1990).

Sierra Simulations & Software: *SIMPLE_1 Version 4 Reference Manual*, Canaan, N.H. (1989).

SysTech, Inc.: *INSIGHT User's Manual*, Indianapolis, Ind. (1985).

White, D. A.: *PCModel User's Guide*, Simulation Software Systems, San Jose, Calif. (1988).