Proceedings of the 1986 Winter Simulation Conference
J. Wilson, J. Henriksen, S. Roberts (eds.)

INTRODUCTION TO DISCRETE-EVENT SIMULATION

Jerry Banks
School of Industrial and Systems Engineering
Georgia Institute of Technology
Atlanta, GA 30332, U.S.A.

John S. Carson II
Carson/Banks & Associates
4279 Roswell Road
Suite 604-173
Atlanta, GA 30342, U.S.A.

ABSTRACT

In this article, we introduce the reader to discrete-event simulation. The concepts of system and model, system state, entities, attributes and delays are defined in the general context of simulation. Using these concepts, event-scheduling, process-interaction, and activity-scanning perspectives are briefly described. To demonstrate the use of the concepts, a discrete system is modeled using the event-scheduling perspective. Simulation languages are classified in terms of the type of system being modeled, the application level, and the perspective taken. The features of a simulation language are discussed. Lastly, basic information is provided about an assortment of discrete-event simulation languages.

1. INTRODUCTION

This tutorial provides an introduction to the main concepts of discrete-event simulation, a simple example illlustrating these concepts, and a discussion of the main classifications and features of the many simulation languages available for discrete-event simulations. The article is introductory in nature and is meant for the reader new to discrete simulation.

Many important aspects of conducting a simulation project are not discussed in this article, for example, choosing probability distributions to represent input random variables, validation and verification, and analysis of the output statistics. (See Banks and Carson [1984] for a discussion of all the important aspects of a simulation study. See Carson [1986] for a practical discussion of validation and verification of simulation models.)

2. CONCEPTS OF DISCRETE-EVENT SIMULATION

In this section, we discuss the concepts of system and model, system state, entities and attributes, events, activities, and delays. Then discrete-event simulation is defined, and contrasted to continuous simulation. Using these concepts, three modeling perspectives have been developed: event-scheduling, process-interaction, and activity-scanning. We show how these concepts are used to model a discrete system using the event-scheduling perspective. The process-interaction and activity scanning perspectives are discussed briefly in Section 3.

2.1. Main Concepts

A model is a representation of a system. The model may be mathematical, descriptive, logical, or some combination of these elements.

System state, or merely state, is a collection of variables that contain all the information necessary to describe the system at any point in time. Thus, state is a function of time. Time itself will be represented by a variable called CLOCK.

An entity is any object or component of the system which requires explicit representation in the model. Examples of entities include parts, products, machines, etc. Each entity may have one or more attributes. The priority and due date of a part are two possible attributes of the part entity. Speed and changeover time for a machine are possible attributes of a machine entity.

An activity is a duration of time of definite length. The length of an activity may be constant, random (specified by a probability distribution), or given as a function of present or past system state. At the instant that an activity begins, its duration is known within the model. In contrast, a delay is a duration of time of indefinite length. At the instant that a delay begins, its duration is not known; it is conditional in nature and will last until the occurrence of some future event whose time of occurrence is presently not known. Two examples of activities are a processing time which is always 4.3 minutes; and the time to failure of a machine, which is defined to be exponentially distributed with mean 2.4 hours. An example of a delay is the waiting time for a mechanic to reach a down machine. The length of the waiting time may depend on complicated system conditions such as the number of other machines that are down, what (if any) other machines break down before the mechanic can reach the machine in question, and the relative importance of this machine. In other words, its duration is not known when it begins.

A discrete-event simulation model is one in which system state changes only at a set of discrete points in time called event times. Between two successive event times, system state does not change. The system state at an event time is called a snapshot. A discrete-event simulation is carried out by increasing simulated time in some manner and updating system state when necessary. Thus, such a simulation proceeds from snapshot to snapshot until it is stopped. In contrast to a discrete-event simulation, in a continuous simulation the system state is allowed to change continuously over time. Continuous simulations are not discussed in this tutorial.

Finally, observations of the simulated system are collected over the course of the simulation and are used to estimate the system performance measures of interest.

2.2 Modeling Perspectives

The three most prevalent modeling perspectives taken by computer simulation languages for discrete-event simulations are event scheduling, process-interaction, and activity scanning. We briefly discuss the first, namely, event scheduling, and give an example. The other two perspectives are discussed in Section 3.

2.3 Event Scheduling

When using the event scheduling perspective, the modeler concentrates on events and their effect on system state. To model a system, the modeler must identify all events, when they can occur, and how system state is affected. The simulation proceeds by arranging all events in chronological order (not all at once, but as the simulation is being carried out) and proceeding from snapshot to snapshot. In this manner, an artifical history (or simulation) of the system is developed.

In simulation languages which take the event scheduling perspective, algorithms and utilities are provided to manage the occurrence of events. Those events which at any given simulated time will occur at some known, definite future simulated time, are listed on the so-called future event list (FEL). Each event on the FEL has an associated event time. The event on the FEL with the smallest event time is called the imminent event, i.e., it is the next event to occur.

EXAMPLE: Consider a single molding machine. We assume that raw material is always available, and that molding time for one part is exactly 1 minute. When one part finishes being molded, the next one begins molding immediately, provided the machine is running. The machine is subject to random failures. The time until failure is 5 minutes with probability 0.6, or 7 minutes with probability 0.4. Repair time is always 2 minutes. We wish to simulate for 20 minutes of operation.

Clearly, this example is an over-simplified version of any real system. We use it merely to illustrate the concepts previously discussed:

System state: The status of the molding machine (up or down)

Entity: Machine at speed 1 part/minute

Events: Failure of the machine
Repair of the machine
Stop the simulation (at time 20)

Activities: Time to failure (random)
Time to repair (constant)

Delays: (None)

First, we need a way to simulate random activity times such as the time-to-failure of the machine. All simulation languages provide a random number generator which will generate a sequence of numbers between 0 and 1. These numbers are sometimes called pseudo-random numbers, because they are supposed to be uniformly and independently distributed on the interval (0,1). For manual simulations, one could use dice, shuffled cards, or a table of random digits appearing in many simulation and statistical textbooks. Next, the original random numbers have to be transformed into the desired random values, in this case either a 5 or a 7. If the random number, call it R, is between 0 and 0.6, then return with a time-to-failure of 5 minutes; but if R is between 0.6 and 1.0, then return with a value of 7 minutes. This technique can be generalized to generate values representative of any probability distribution. For the purposes of this example, we assume that random numbers have been generated and transformed into the following values:

Successive times-to-failure: 5 7 7 ...

In a longer sequence, the 5's and 7's would appear in random order with about 60% fives and 40% sevens.

Next we need a way to generate event times during the course of the simulation. As a general rule, event times are not all generated at simulated time 0, but rather are generated only as needed. In this simulation, the event time for the next failure of the machine will be generated at the simulated instant when the machine has just been repaired and is beginning a runtime, by the formula:

event time for next failure event =
CLOCK + time-to-failure,

where CLOCK, the current simulation time, is the instant of repair, and time-to-failure is generated at random (and will be 5 or 7).

Before beginning the simulation, we need to make assumptions concerning initial conditions. We assume that at simulated time 0, the machine is freshly repaired and just beginning a runtime.

Finally, we set up a standard simulation table to display the simulation snapshots. (See Table 1.) The left-most column is labelled "CLOCK" and represents current simulated time. It is initialized to a value of zero. The second column is labelled system state, and in general would be a list of variables; in this example, it is merely machine status (up or down). The third column is labelled "FEL" for future event list. The right-most column is labelled "Statistics"; in this example, only one statistic is computed, namely, number of parts produced from time 0 to the current simulated time.

Table 1: Simulation Table for the Single Machine Problem

CLOCK	System State	FEL	Statistics
	Machine status		No. parts produced
0	up	Next failure at 0+5 Stop at 20	0
5	down	Next repair at 5+2 Stop at 20	5
7	up	Next failure at 7+7 Stop at 20	5
14	down	Next repair at 14+2 Stop at 20	12
16	up	Stop at 20 Next failure at 16+7	12
20	up	Next failure at 23	16

Each row of the table represents a snapshot of the system at the simulated time given by CLOCK. For example, the first row, with CLOCK set to 0, represent initialization; it gives machine status as up, the "Stop the simulation event" with event time 20, the first failure to occur at time 5, and number of parts produced initialized to zero. The simulation proceeds by moving from one snapshot to the next until the "stop" event is executed. Note that at any given simulated time, the FEL consists of a list of all known future events which have been generated at the current or some past simulated time. Thus the "Stop at 20" event is carried along until time 16, at which time it becomes the imminent event.

In interpreting the simulation table, note that system state does not change between two successive snapshots. For example, machine status is up from time 0 to time 5, while machine status is down from time 5 to time 7. Finally, the event times are written as "a + b" simply to illustrate how they are computed.

3. SIMULATION LANGUAGES

In this section we classify simulation languages on the basis of the type of system being modeled, the application of the simulation language for a general or special purpose, and the modeling perspective taken. Next, we describe many features that a simulation language may possess. Lastly, we provide some basic informtion about the various discrete-event languages that are available.

3.1. Level of Application

Simulation languages can be classified at three different levels:

1. System
2. Application
3. Structural

3.1.1. System Level

A language can be classified on the basis of the type of system modeled. The two types of systems that are generally recognized are discrete and continuous.

In continuous systems, system state can change continuously over time. Continuous simulation models employ differential equations or difference equations which describe the rate of change of the state variables over time. These equations are usually solved at specified increments of time to determine the current value of all state variables.

In discrete systems, the state variables change only at discrete points in time. These times were referred to previously as event times. Systems involving waiting lines, inventory operations, many manufacturing operations, and material handling systems are usually regarded as discrete systems.

3.1.2. Application Level

Discrete-event simulation software can be classifed as special purpose or general purpose. Special purpose simulation languages are designed to model specific environments. Special purpose languages offer speedier model development. An example application of a special purpose language is the modeling of a batch manufacturing facility which uses conveyors and AGVs as the material handling devices.

A special purpose language which requires only the definition of the environment being simulated is called a "simulator." This is in contrast to general purpose simulation software products which are commonly called a "simulation language." In recent years, general purpose simulation languages have been adding modules for conveniently modeling various material handling aspects. Thus, some general purpose languages are adopting the features of simulators.

3.1.3. Structural Level

The structural level is concerned with the modeling perspective taken by the discrete-event language. Classification at this level is meaningful for general purpose simulation languages only, as the simulators can be considered as data driven. The three most prevalent orientations are as follows:

Event Scheduling: In this perspective, a system being modeled is viewed as consisting of a number of possible events at which state changes take place. The modeler defines the events, and develops the program and logic associated with each event. The event scheduling perspective was discussed previously.

Process Interaction: Simulation languages that adopt this perspective allow a modeler to represent a system as a set of processes. A part flowing through several workstations is an example of a process. Another name used to represent process interaction simulation languages is network simulation languages. In virtually all network simulations, a block diagram or network is drawn first, followed by statements which represent the blocks.

Activity Scanning: In this perspective, the modeler defines the conditions necessary to start and end each activity and delay in the system. As mentioned previously, an activity is a duration of specified length and a delay is a duration of unspecified length. Simulation time is advanced in equal increments, and if the conditions are appropriate, an activity will be started or terminated. Activity scanning is not popular in the United States.

3.2. Features of a Simulation Language.

There are numerous features that a modeler may expect from a simulation language. A few of the prominent features are as follows:

1. Graphics/animation
2. Database management
3. Material handling/manufacturing
4. Interactive debugging
5. Support/documentation
6. Microcomputer version

These features will be discussed briefly in the following paragraphs:

3.2.1. Output Graphics.

Output graphics can be divided into two categories, static and dynamic. Static graphics are those in which the picture is a constant, i.e, the picture is not changing over time. Static graphics may take the form of pie charts, bar charts, histograms, line graphs, and plots. Dynamic graphics are those in which the picture depicts a variation in the state of the system, i.e., the picture is changing over time. Dynamic graphics may be divided into statistics/states and animation. Dynamic statistics/states take the same forms as the static graphic pictures except they change over time as the statistics and states change. Animated graphics look like the system being simulated, in varying degrees of sophistication and detail. Animations can be further divided into two categories. The first category are those which use character graphics. The second category utilizes pixel (picture element) graphics. Cartoon animation is accomplished with pixel graphics.

3.2.2. Database Management.

Some realistic simulation models require significant amounts of problem specific input data. The amount of data may increase if there are various input scenarios which are to be simulated. Usually, many replications of a simulation are conducted, and the outputs and/or their summaries are to be saved. Combine these data management tasks with the analysis of input data to determine frequency distributions and the storage of externally loaded sequential files, and the need for a database management system becomes apparent.

3.2.3. Material Handling/Manufacturing

In realistic manufacturing simulations, the material handling aspects are critical. Using a general purpose simulation language without special features for material handling causes difficulties in the modeling task. A very lengthy program is necessitated. Modules that provide the capability to model conveyors, cranes, AGVs, towlines, etc. can reduce model development time substantially and also reduce the debugging effort.

3.2.4. Interactive Debugging

Traces are a common method employed for debugging a simulation model. However, debugging is greatly simplified if the modeler can control the simulation, one step at a time, and can access simulation output during the stepping process.

3.2.5. Support/Documentation

Users should consider the support provided by a vendor when selecting a simulation language. Adequate support implies clear and understandable documentation. The vendor must have staff persons who are readily available to answer user questions. The best way to determine the quality of the support offered by a vendor is to ask other users of the software about their experiences.

3.2.6. Micro Versus Mainframe

The number of simulation languages available on the microcomputer is large and growing. The question often arises concerning the selection of a simulation language, i.e. "Should we purchase the microcomputer or mainframe version?". Small simulation models can be run on the microcomputer, but large models are very time consuming using these machines. The problem is compounded when numerous runs must be made to attain confidence in the output that is generated. In defense of the microcomputer, the user is not at the mercy of the system operators who take the mainframe down for repairs, change operating systems, lock files, and so on. Some simulation languages have versions which are compatible on both levels.

3.3. Additional Factors

In selecting a simulation language, the potential purchaser must consider a number of factors in addition to those mentioned above. Briefly, these factors include the following:

Syntax: The syntax used in the simulation language should be easy to decipher and not require an excessive amount of innocuous information from the user.

Structural Modularity: Simulation languages should allow development in modules such as model description, experimental conditions, output report requirements, etc.

Modeling Flexibility: A wide range of problems can be simulated when the language allows event scheduling, process interaction, and some combination of the two perspectives.

Modeling Conciseness: Whether using the event scheduling or process interaction approach, powerful blocks/nodes or subroutines can enable timely development of models.

Statistics Generation: Comprehensive statistics should be provided by the simulation software, or the statistics should be easy to develop.

Cost: The cost of the various simulation packages varies widely from a few hundred dollars for a microcomputer version to over $50,000 for a material handling software package with animation.

3.4. Basic Information on Discrete-Event Simulation Languages.

There are many simulation languages that are available, a number of which are being described during this Winter Simulation Conference and many of which are being exhibited. In Table 2 we offer a concise guide to the features of some of the more popular of these languages. This table is a modification of a similar one in a recent series on simulation (Haider and Banks, 1986). Since there are so many discrete simulation languages, it is not possible to describe all of them. We apologize to those vendors whose language is not represented in Table 2.

Note in Table 2 that references are given in the right hand side. These references offer additional information about the languages. Some further explanation of Table 2 is warranted as follows:

Letters G and S denote general purpose or special purpose software, respectively. If the software is special purpose then NA for not applicable is indicated in the orientation column. Otherwise, the perspective is indicated as E for event scheduling, N for network or process interaction, and U for user written process interaction.

Interactive debugging capability in SLAM II is available through TESS for its mainframe version; and in SIMSCRIPT II.5, it is available for the PC version.

Table 2: Basic Information on Simulation Software

Simulation Software	General Purpose Special Purpose	Perspective	Interactive Debugging	Material Handling Feature	Animation	PC/ Mainframe Version	Built in Interface With a Data Base	References
AutoMod	S	NA		x	x(AutoGram)	M		AutoMod User's Manual
GPSS V	G	N				M		Gordon
GPSS/H	G	N	x		x (TESS)	M	x (TESS)	Henriksen & Crain
GPSS/PC	G	N	x			P		GPSS/PC User Manual
MAP/1	S	NA		x		M	x (TESS)	Rolston & Miner
PCModel	G	U	x		x	P		White
RESQ	G	N	x			M		Chow, MacNair, & Sauer
SEE WHY	G	E	x		x	P/M		Tom
SIMAN	G	E, N	x	x	x (Cinema)	P/M		Pegden
SIMFACTORY	S	NA		x	x	P		Not available
SIMPLE_1	G	N	x		x	P		Cobbin
SIMSCRIPT II.5	G	E, U	x (PC only)		x (Simanimation)	P/M		Russell
SLAM II	G	E, N	x (TESS)	x (MHE)	x (TESS)	P/M	x (TESS)	O'Reilly
XCELL	S	NA	x		x	P		Conway, Maxwell, & Vorona

Under the column entitled "Material Handling Feature", an "x" indicates that the capability to model some aspects of material handling systems exists in the software. SLAM II offers material handling capabilities in a special extension called "MHE."

In some simulation software, animation is available as a special module. For example, AutoGram provides animation for AutoMod. Animation for SIMAN and SIMSCRIPT II.5 is currently available for the PC versions of the language. In SLAM II, animation is available only for the mainframe version.

The next column indicates whether the software is for the mainframe (M) or the personal computer (P) or both (P/M). Although the distinction between the personal computer and the mainframe is clear at present, the difference between the two is narrowing. Machines of the size of an IBM PC AT and below are personal computers to us, and machines in the IBM 43xx class and larger are mainframes. Although some simulation software falls between these two extremes, the purpose of the information in this column is to indicate the relative nature of the hardware requirements.

Interfacing with a database management system is available in MAP/1, SLAM II and GPSS/H using TESS. TESS is available only for selected minicomputers and mainframes of SLAM II.

The complete references indicated in the last column of Table 1 are given at the end of this tutorial.

REFERENCES

AutoMod User's Manual (1986). AutoSimulations, Inc. Bountiful, Utah.

Banks, J. and Carson, J. S. (1984). Discrete-Event System Simulation. Prentice-Hall, New Jersey

Carson, J. S. (1986). Convincing Users of A Model's Validity. Industrial Engineering 18, 6.

Chow, W., MacNair, E. A. and Sauer, C. H. (1985). Analysis of Manufacturing System by Research Queueing Package. IBM Journal of Research and Development 30, 4.

Cobbin, P. (1986). SIMPLE_1: A Simulation Environment for the IBM PC. In: Proceedings, (Barnett, C., ed.). SCS Conference on Modeling and Simulation on Microcomputers.

Conway, R. W., Maxwell, W. L. and Vorona, S. L. (1986). Users Guide to XCELL - Factory Modeling System. Scientific Press, California.

Gordon, G. (1975). The Applications of GPSS V to Discrete Systems Simulation. Prentice-Hall, New Jersey.

GPSS/PC User Manual Version 1.1 (1985). Minuteman Software. Stow, Massachusetts.

Haider, S. W. and J. Banks (1986). Simulation Software Products for Analyzing Manufacturing Systems. Industrial Engineering. 18, 7.

Henriksen, J. D. and Crain, R. C. (1983). GPSS/H User's Manual. Wolverine Software Corporation. Annandale, Virginia.

O'Reilly, J. J. (1985). SLAM II: A Tutorial. In: Proceedings of the 1985 Winter Simulation Conference (D. T. Gantz, S. L. Solomon, and G. C. Blais, eds.). Institute of Electrical and Electronics Engineers, San Francisco, California.

Pegden, C. D. (1985). Introduction to Siman. In: Proceedings of the 1985 Winter Simulation Conference (D. T. Gantz, S. L. Solomon, and G. C. Blais, eds.). Institute of Electrical and Electronics Engineers, San Francisco, California.

Rolston, L. J. and Miner, R. J. (1985). MAP/1 Tutorial. Proceedings of the 1985 Winter Simulation Conference (D. T. Gantz, S. L. Solomon, and G. C. Blais, eds.) Institute of Electrical and Electronics Engineers, San Francisco, California.

Russell, E. C. (1985). SIMSCRIPT II. Tutorial. Proceedings of the 1985 Winter Simulation Conference (D. T. Gantz, S. L. Solomon, and G. C. Blais, eds.) Institute of Electrical and Electronics Engineers, San Francisco, California.

Tom, L. (1985). See Why. Simulation. 45, 2.

White, D. A. (1985). PCModel - Personal Computer Screen Graphics Modeling System User's Guide. Simulation Software Systems. San Jose, California.

AUTHORS' BIOGRAPHIES

JERRY BANKS is Associate Professor of Industrial and Systems Engineering. His area of interest is simulation languages and modeling. He is the author of numerous articles and books. His recent books were Discrete-Event System Simulation, co-authored with John S. Carson, II, published by Prentice-Hall in 1984, Handbook of Tables and Graphs for the Industrial Engineer and Manager, co-authored with Russell G. Heikes, published by Reston Publishing Co., a division of Prentice-Hall also appearing in 1984, and IBM PC Applications for the Engineer and Manager, co-authored with J. P. Spoerer and R. L. Collins, and published by Reston in 1986. Dr. Banks received his Ph.D in Engineering from the Oklahoma State University in 1966. He is an active member of many technical organizations. He serves as IIE's representative to the Board of the Winter Simulation Conference.

Jerry Banks
School of Industrial and Systems Engineering
Georgia Institute of Technology
Atlanta, GA 30332, U.S.A.
(404) 894-2312

JOHN S. CARSON II is a simulation consultant to a number of manufacturing and service industries. He has served on the faculties of the Georgia Institute of Technology and the University of Florida. He is coauthor, with Jerry Banks, of the text Discrete-Event Systems Simulation. He is a member of IIE, ORSA and TIMS. Dr. Carson earned his BA in mathematics from the University of Virginia

and his MS in mathematics and his PhD (1978) in engineering from the University of Wisconsin at Madison.

John S. Carson II
Carson/Banks & Associates
4279 Roswell Road
Suite 604-173
Atlanta, GA 30342, U.S.A.
(404) 971-1256