Proceedings of the 1992 Winter Simulation Conference
ed. J. J. Swain, D. Goldsman, R. C. Crain, and J. R. Wilson

MODELLING

John S. Carson

Carson/Banks & Associates, Inc.
4665 Lower Roswell Road, #140
Marietta, GA 30068

ABSTRACT

This paper provides a language-independent introduction to the major simulation modelling world views for discrete-event systems simulation. A world view is the modeling framework which a modeler uses to represent a system and its behavior. The main terminology and concepts include systems and models, system state variables, entities and their attributes, lists, resources, events, activities and delays. These concepts are covered from the perspective of the modeler. Finally, in the tutorial, we will attempt to make these ideas concrete through the use of a number of examples and exercises.

1 INTRODUCTION

This paper introduces the major world views of simulation modelling from the perspective of a person about to begin development of his or her first model using a discrete-event simulation language. The purpose is to give modelers a broader perspective, and to increase understanding of the world views built into the major commercially available simulation languages. Both the article and the tutorial are introductory in nature and are meant for the reader new to discrete-event simulation.

We discuss the event-scheduling world view, historically the first approach to discrete-event systems simulation. Then we discuss the process interaction and transaction flow world views, and their implementation through process procedures, block diagrams, or arc/node representations.

In the tutorial, we will illustrate how a modeler may take more than one view of a given problem, leading to a number of possible models of one system. Through an example, we illustrate how a simple problem can be modelled in more than one way, depending both on the world view of the language being used as well as the view taken by the modeler.

Many important issues of simulation are not discussed in this paper, but are discussed in other papers in the introductory series.

2 BASIC CONCEPTS

There are a number of concepts underlying all the major world views in discrete-event simulation modelling. These include system and model, system state, entities and attributes, lists, event and event notices, resources, and activities and delays.

2.1 System and Model

A model is simply a representation of a system. The system should be defined with clear boundaries between the outside world and the portion of the world being simulated. Generally, the system components are simulated in varying degrees of detail, depending upon their perceived importance with respect to the modelling objectives, while the outside world is simulated only to the extent that specific events impinge upon the system. A simple example is random arrivals (from the outside world) that place demands upon system resources.

A discrete-event simulation model is a specific type of model that may be contrasted to other types such as mathematical models, descriptive models, statistical models, and input-output models. A discrete-event model attempts to represent the components of a system and their interactions to a level of detail sufficient to meet the objectives of the study and to answer the questions appropriate to those objectives.

With their detailed representation of system components, discrete-event models stand in contrast to some mathematical, statistical and input-output models that represent a system's inputs and outputs

explicitly, but attempt to represent the inner workings by a mathematical or statistical relationship derived empirically or on some other ad hoc basis, not by a detailed representation of the actual inner workings. Some mathematical models, for example those from physics, are based on a theory and are not merely empirical as are many statistical models based solely on data.

Discrete-event models are dynamic with respect to time. In other words, time plays a crucial role in the sense that the variables defining system state are functions of time. Many mathematical, statistical and spreadsheet models are static; they represent a system's state at a fixed point in time.

Discrete-event models can also be differentiated from continuous models, based on the nature of the variables needed to track system state. The system state variables in a discrete-event model remain constant over intervals of time and change value only at certain well-defined epochs (points in time) called event times. On the other hand, continuous models have system state variables defined by differential or difference equations and thus its variables change continuously over time. Some models are mixed continuous and discrete, simply due to the nature of the system being modeled or because of modelling efficiencies; that is, in some circumstances, discrete systems can be efficiently modeled with continuous models, and continuous systems with discrete models.

We give a more precise definition of discrete-event modelling after further discussion of system state and events.

2.2 System State Variables

As implied in the previous section, system state variables are the collection of all variables needed to define system state to a level of detail sufficient to meet the project's objectives. Determining those variables is matter of art and experience as much as science. Fortunately, for the most part, the act of modelling, of putting a model together, will bring to the fore any initial oversights in identifying system state variables.

2.3 Events and Event Notices

An event in a model corresponds to a happening in the real system that changes or potentially changes the state of the system. In a model, future events are modelled by making assumptions or using collected data regarding the times of occurrences of these events. These so-called event times are recorded in event notices and managed by the underlying simulation software system in such a way (discussed later) as to cause events to occur at proper times in the model. Note that an event occurs at an instant of time; it does not have a duration.

Examples: an arrival to a system; the completion of service at a particular service center; a breakdown of a machine.

In Section 2.6, we distinguish two types of event, the primary or unconditional event, and the secondary or conditional event.

2.4 Entities, Attributes and Lists

An entity in a model represents some object or element of the system that needs to be explicitly modeled. An entity may be dynamic in nature in the sense that it "moves" through the system, or it may be static in the sense that it services other entities.

Examples: a customer in a queueing system (dynamic entity); a server in a service system (dynamic or static depending on how modeled)

An entity may have one or more attributes, or properties, that belong to that entity. There may be many entities of a particular entity class, all having the same attributes but each having its own value for an attribute.

Examples: a customer may have a demand, a due date, and a priority; a server may have a service rate and a schedule.

Typically, entities are managed by attaching them to resources (discussed next in section 2.5) that provide service to them, by attaching them to event notices (thus suspending their activity until some future time), or by placing them into an ordered list. Lists of entities are used to represent queues, places in a system where entities are forced to wait due to scarce resources or other system conditions. In the simulation literature (and in simulation languages), lists are variously referred to as queues, files, chains or some other similar term. A list may have a FIFO ordering (first-in, first-out), so that entities are joined to the back of the list and removed from the front of the list. In fact, a list may be FIFO, LIFO (last-in, first-out), or sequenced by a specified attribute of an entity. For example, an entity representing orders may be sequenced on a list from smallest to largest due date, representing the priority in which orders are to be handled.

2.5 Resources

A resource is a type of entity that provides service to other entities. A resource may have a capacity or a

number of servers in parallel. Typically, a dynamic entity may request one or more units of a resource. If denied, the entity joins a queue to wait or takes some other specified action; if not denied, the entity captures the units of the resource for a duration of time and eventually releases the units of the resource.

Typically at minimum, a single-unit resource has a status attribute representing busy and idle states; more generally, a multi-unit resource has an attribute for number of captured units. In many language implementations of these concepts, a resource may have additional attributes representing other possible states. These may have a name such as availability (two values: available and unavailable).

Examples: a single server in a queueing system; a bank teller; a machine; a grouping of two automated teller machines.

A resource is manipulated by two actions, here called CAPTURE and FREE, and typified by the following pseudo-code referring to a resource called MACHINE:

```
CAPTURE 1 UNIT OF MACHINE
WAIT FOR 10 MINUTES
FREE 1 UNIT OF MACHINE
```

2.6 Activities and Delays

An activity is a definite duration of time that is explicitly defined by the modeler. It may be a constant duration, a random duration (definition based on a random number generator), a formula, or a value from an input data file; it may be computed by any means within the model. The key is that it is definite and known to the model at the instant it begins. The nature of an activity is expressed by the following pseudo-code:

```
WAIT FOR x TIME UNITS
```

where x is explicitly defined by the modeler and is either independent of past, current and future system state, or is at most a computable function of past and current system state. As we will see, an activity time usually "holds" a dynamic entity for its duration.

Examples: service times, time to failure of a machine or component, time between arrivals

Some typical examples include:

```
WAIT FOR 3 MINUTES
```

where MINUTES refers to simulated time units, or

```
WAIT FOR EXPONENTIAL(10.2) MINUTES
```

where the wait is for a random amount of time randomly generated as a sample from an exponential distribution having mean 10.2 minutes.

A delay is an indefinite duration of time that is caused by some combination of system conditions. The nature of a delay is expressed by the pseudo-code:

```
WAIT UNTIL CONDITION y IS TRUE
```

meaning that a process or entity is "held" until some specific system condition becomes true. Typically in complex real-world models, a delay is a complex function of current and future system states and cannot be computed ahead of time. Only the dynamic "working out" of the simulation logic can determine the duration of a delay. Specific examples of delays include:

```
WAIT IN A QUEUE FOR YOUR TURN TO BE SERVICED
```

or,

```
WAIT UNTIL LEVEL IN TANK IS < 2400 GALLONS
```

or,

```
WAIT UNTIL MACHINE A IS AVAILABLE AND IDLE
```

All discrete-event simulations contain activities, else time does not advance in the simulation. Virtually all such models have entities queueing, resulting in delays to these entities.

The beginning and end of activities and delays are events. Primary or non-conditional events are ones that occur at the end of an activity time. Secondary or conditional events are ones that occur as a consequence of primary events occurring during a particular system state or that occur at the end of a delay time.

For example, if arrivals to a system are defined in terms of a constant interarrival time, an arrival becomes a primary event. On the other hand, when a customer in a queue begins service is a secondary or conditional event.

2.7 Discrete-event simulation model

With the above definitions and concepts in mind, we can now give a more complete definition of a discrete-event model, and a hint for how such a simulation is carried out.

A discrete-event model is one in which the system state variables change only at those discrete points in time at which events occur. Events occur from time to time as a consequence of activity times and delays. Entities may compete for system resources, possibly joining queues while waiting for a free resource. Activity and delay times may "hold" entities for durations of time.

Such a discrete-event model is carried out over time (or "run", not solved) by a mechanism that moves simulated time forward from current event to next event, updating system state at each event, and

freeing and capturing system resources. The underlying mechanism involves event "scheduling" and event execution in proper order with respect to simulated time.

2.8 World views

The representation of a discrete-event model and the implementation of the event-scheduling mechanism is a function of the world view. We will discuss the major world view used by the popular and time-tested major languages in use in the USA today. These world views are event orientation, process interaction, and transaction flow.

2.8.1 Event oriented world view

When using the event oriented world view, a modeler first identifies all events and then writes event routines that completely define the consequence of each event. Recall that an event occurs in an instant of time, so that an event routine occurs in zero simulated time. The consequences of an event may include:

- One or more system state variable may change value
- One or more conditional events may be triggered to occur now, as a combined consequence of this event and current system state
- One or more primary events may be "scheduled" to occur at some future simulation time

At time zero during model initialization, the modeler must do the following:

- Initialize all system state variables
- Initialize all resource capacities and states
- Generate and schedule at least one primary event

Scheduling an event consists of, first, creating an event notice, that is, a record that contains event type and (future) event time, and secondly, placing the event notice on a special list of event notices called the event list or future event list. Event notices on the event list are ranked from smallest to largest event time. The event notice at the top of the event list defines the imminent event or the event to occur next in simulated time.

Every simulation has one stopping event, the event that designates simulation completion. It may be a primary event, as for example, "stop the simulation after exactly 8 simulated hours". Or it may be a conditional event, such as "stop the simulation after 100 customers have completed service".

The event oriented world view is implemented by the event scheduling algorithm:

1. Initialize the model, as described above.
2. Remove the event notice for the imminent event from the event list (event notice with smallest event time.)
3. Advance the simulation clock to the event time of the imminent event.
4. Execute the imminent event.
5. If the simulation is not complete, go to step 2.

Otherwise, stop the simulation and report results.

In Step 4, one of the event routines is executed. In this manner a simulation is carried forward in time, from event to event, until the stopping event occurs.

Event oriented simulations may contain entities that are created upon the occurrence of certain events, whose attributes change when other events occur, and which capture resources and join queues. However, the point of view is that of an event and to envision the "flow" of an entity through the system takes imagination and an understanding of the entity's interaction with the possibly many events affecting it.

Event scheduling simulations are traditionally implemented in a general purpose programming language such as FORTRAN, Pascal, or C. There are a number of commercially available libraries of routines in FORTRAN and other languages to facilitate the programming of event oriented simulation models. These libraries of event-scheduling routines implement tasks common to all event oriented models, such as management of the event list, management of lists of entities (queues), statistics collection for output reporting, and management of resources.

A number of commercially available simulation languages that emphasize a process interaction or transaction flow world view also have an (optional) built-in event-scheduling world view, and allow a model to be a mix of process and event oriented.

2.8.2 Process interaction world view

The process interaction world view provides a way to represent a system's behavior from the point of view of the dynamic entities moving through the system. A process is a time-ordered sequence of events, activities and delays that describe the flow of a dynamic entity through a system.

Languages implementing a process interaction world view require the modeler to write process routines, which are quite different from event routines. While event routines occur in zero time, process routines may contain both activities and delays. Process routines are not possible in ordinary

programming language such as FORTRAN and Pascal; they require special mechanisms for interrupting and suspending the execution of a routine, and resuming execution at a later simulated time under the control of an internal event scheduler. (Internally but hidden from the view of the modeler, there are still events being scheduled and an event list being manipulated.)

This world view is call process interaction, because active processes may interact as they compete for limited resources and in other ways. This process interaction is handled essentially automatically by languages that utilize the process world view.

As an example, consider customers arriving at a bank and queueing for service from the three tellers on duty. Here is some pseudo-code representing a customer process:

```
  CUSTOMER PROCESS:
     CUSTOMER ARRIVES              (arrival event)
'A'  IF ALL UNITS OF TELLER        (entity joins list)
       RESOURCE ARE BUSY,
       JOIN TELLER QUEUE
     ELSE                          (capture
                                    resource)
'C'    CAPTURE ONE TELLER          (service time)
       WAIT FOR x TIME             (free resource)
       UNITS
       FREE TELLER
     ENDIF                         (departure event)
     CUSTOMER DEPARTS
```

Actually, to have a complete model requires at least two other abstract processes, or the customer entity must take on additional duties that do not, strictly speaking, correspond to customer activity in the real world. The two processes referred to are one to create additional future arrivals and the second to remove a CUSTOMER entity from the list called TELLER-QUEUE whenever a service completion event occurs.

The arrival process can be modelled in several different ways. One is to envision the line "CUSTOMER ARRIVES" above as a subsidiary process that is used to "generate" future arrivals, as illustrated by the following pseudo-code:

```
  CUSTOMER ARRIVAL PROCESS:
'START' GENERATE T = INTERARRIVAL
          TIME
        WAIT FOR T TIME UNITS
        CREATE NEW CUSTOMER ENTITY
        SEND NEW CUSTOMER ENTITY TO
          CUSTOMER PROCESS, LOCATION
          'A'
        GO TO 'START'
```

Many process interaction languages implement such an entity arrival process in one statement in the language, so that a modeler does not have to explicitly program such processes.

We also need to create an abstract process, or extend the customer entity's duties, to bring the first customer off the queue whenever a teller becomes idle, perhaps as follows:

```
  FREE TELLER PROCESS:
    IF TELLER-QUEUE IS NON-EMPTY,
      REMOVE FIRST CUSTOMER ENTITY
       FROM TELLER-QUEUE
      SEND CUSTOMER IMMEDIATELY TO
       LOCATION 'C'
    ENDIF
```

Note that the "FREE TELLER PROCESS" occurs in zero time; there are no activities or delays. In effect, it is an event. (A zero time process procedure is equivalent to an event routine.) Many simulation languages implement the equivalent of the "FREE TELLER PROCESS" automatically, so the modeler does not need to explicitly address the handling of the customer queue.

The process world view is the most popular world view in terms of commercially available simulation languages in the USA.

2.8.3 Transaction flow world view

A transaction flow world view is a special case of the (possibly) more general process interaction world view. A transaction is nothing but another name for a dynamic entity that flows through a system.

In the transaction world view, transactions are viewed as flowing through a block diagram or a network of arcs and nodes. In the block diagram framework, epitomized by GPSS$_{TM}$ and also adopted by SIMAN$_{TM}$, blocks represent either immediate events, activities or delays. Languages such as GPSS have a collection of blocks to represent capture and release of resources, modeler manipulation of lists of transactions, and activities (explicitly defined passages of time). The block diagram, similar to a process flow diagram, is a set of blocks connected by arrows, representing the transaction process.

Other simulation languages, such as SLAM II$_{TM}$ and SLAMSystem$_{TM}$, adopt a node/arc point of view, with arcs representing activities and nodes representing all other events and actions. The node/arc point of view is similar in philosophy to the block diagram point of view.

In the transaction flow world view, process code similar to that in the preceding section is viewed as being executed by the entity or transaction as it moves from statement (block, node or arc) to

statement (next block, node or arc). For some implementations of this world view, this makes the transaction flow world view somewhat more restrictive than the process interaction world view.

3 SUMMARY

In the tutorial, we will give one or more examples and exercises to illustrate the concepts discussed in this paper. With one example, we will provide a model from a number of different perspectives, illustrating how two people can develop quite different though equally accurate models of the same system.

REFERENCES

Banks, J. and J.S. Carson, 1984. Discrete-Event System Simulation. Prentice-Hall.

Law, A.M. and W.D. Kelton, 1992. Simulation Modeling and Analysis, 2nd Edition. McGraw-Hill, New York.

AUTHOR BIOGRAPHY

JOHN S. CARSON is President of Carson/Banks & Associates, Inc., a simulation consulting firm. He has developed a number of large-scale simulation models in a variety of application areas, including manufacturing, material handling, warehousing, distribution, transportation and rapid transit, medical delivery systems, and reservations systems. He has served on the faculties of the Georgia Institute of Technology and the University of Florida. With Jerry Banks, he is the co-author of the widely-used text Discrete-Event Systems Simulation (Prentice-Hall, 1984). He is a member of IIE, ORSA, and TIMS. Dr. Carson received his Ph.D. in Industrial Engineering and Operations Research from the University of Wisconsin at Madison (1978).