Proceedings of the 1994 Winter Simulation Conference
ed. J. D. Tew, S. Manivannan, D. A. Sadowski, and A. F. Seila

INSIDE SIMULATION SOFTWARE: HOW IT WORKS AND WHY IT MATTERS

Thomas J. Schriber

Computer and Information Systems
The University of Michigan
Ann Arbor, Michigan 48109-1234 U.S.A.

Daniel T. Brunner

System Flow
6366 Guilford Avenue, Suite 310
Indianapolis, Indiana 46220 U.S.A.

ABSTRACT

This paper provides beginning and intermediate simulation practitioners and interested simulation consumers with a grounding in how discrete-event simulation software works. This is done by describing alternative entity states, the use of lists to organize entities in the various states and the use of algorithms to manage these lists and manipulate entities during a simulation. This general treatment, which should benefit all simulation practitioners, is then discussed in terms of GPSS/H, ProModel and SIMAN. The focus is on understanding the underlying mechanisms of simulation. Such understanding will help practitioners build models more confidently, use model verification tools more thoroughly, and model complex system logic precisely as intended.

1 INTRODUCTION

1.1 Background

A "black box" approach is often taken in teaching and learning discrete-event simulation software. The characteristics of the tools provided by the software are studied and examples for use of the tools are given, but the underlying foundation that is the basis for these tools is touched on only briefly or is even ignored. We believe that simulation practitioners should ideally know the characteristics of the tools at their disposal *and* should understand how these tools fit into and are derived from the logical underpinnings of discrete-event simulation. Practitioners who can thus "see through two lenses" are in a position to build and verify models faster and use simulation tools more imaginatively.

Taking the point of view of the practitioner into account, our objective here is to briefly present the logical underpinnings of discrete-event simulation and relate this material to the major tools provided in three instances of discrete-event simulation software.

1.2 Structure of the Paper

In Section 2 of the paper we briefly comment on simulation experiments, bringing the focus down to single simulation runs and then homing in on the start-to-finish internal processing that takes place during a run at a single instant of simulated time. It is at single instants of simulated time that simulation software has to handle simultaneous events serially, and it is this handling with which much of the paper deals.

Section 3 describes the transaction-flow world view and how it differs from conventional procedural programming. The concept of "units of traffic" is reviewed and then, to lay the groundwork for examination of specific algorithms, the alternative states through which units of traffic migrate as a simulation proceeds are introduced.

Next, in Section 4, we introduce the list data structures used by simulation software to manage individual units of traffic. Roughly speaking the *types* of lists available in each simulation software tool are the same. The list types generally correspond to the various states introduced in Section 3.

This leads to a Section 5 discussion of how competition among units of traffic for scarce system resources is managed in each of the three tools. The goal in this section is to describe the general structure of the software and its primary algorithms. Some of the finer details are deferred to Section 6, where we turn to specific examples of conditions that one might want to model, and describe how the approach would differ among the software tools under consideration.

1.3 Terminology and Conventions

Throughout this paper we use terms that we define as well as terms reserved by the developers of a particular simulation tool and/or its documentation. Terms we define are *italicized* on first use and are expressed in lower-case normal type subsequently. Terms that are tool-specific are Capitalized or, where appropriate, are spelled out in ALL CAPS.

We also frequently use a "part/machine" metaphor to illustrate specific points. In a classical job-shop manufacturing model, a part waits with other parts to use a machine. When it is that part's turn to use the machine and the machine is ready to be used, the part undergoes processing for a particular (possibly randomly sampled)

time. This metaphor is easily transposed to many other types of manufacturing and non-manufacturing models.

2 OVERVIEW OF MODEL EXECUTION

2.1 Experiments, Replications, and Runs

Conducting a simulation project includes carrying out one or more *experiments*. Experiments are differentiated by the introduction of one or more alternatives in the underlying logic or data used in a simulation model. An alternate part sequencing rule might be tried, for example, or the quantity of constrained machines might be varied.

Each experiment consists of one or more *replications*. A replication is a simulation that uses given model logic and data but a different set of random numbers, and so produces different statistical results that can then be analyzed across a set of replications.

Each replication consists of initializing the model, running it for a period of simulated time or until some condition is met, and reporting results. We call the "running it" phase a *run*.

2.2 Inside a Run

During a run the simulation *clock* tracks the passage of simulated time. The clock is an internally managed stored data value that changes during the run. The clock advances in discrete steps. Generally the steps are not of equal size. (*Fixed time increment* simulations are not considered here). When the clock changes, its new value is set equal to the time of the next scheduled *event*.

Computations and logic within a run are executed at a series of discrete instants of simulated time. When all computations and logic that can occur at a particular time have been completed, the clock is *updated* (advanced).

The execution of a run thus takes the form of a two-phase loop: "execute all possible events at the current time" followed by "advance the clock," repeated indefinitely. We call the two phases the *Entity Movement Phase* (EMP) and the *Clock Update Phase* (CUP), respectively. During either phase a run-ending condition may become true, causing the run to conclude.

Each EMP requires a variable amount of *computer time* to complete. The EMP unfolds according to a set of rules that are different for each software tool. Although a given EMP takes place at a single instant of simulated time, the underlying software has much work to do to be sure all pending and newly generated events get processed correctly during that time instant.

3 TRANSACTION-FLOW WORLD VIEW

3.1 Entities

Most discrete event simulation software operates on what has been labeled the *transaction flow world view*. By this we mean the person building the model (whom we'll call the *modeler*) views the simulated system as a network of *operations* — conceptually similar to a flow chart — through which individual units of traffic flow (move). The units are called *transactions* or *entities*; we'll use "entities" as our generic term.

In real-world terms, the concept of an entity often closely corresponds to physical constructs. In a highway traffic model, a vehicle is an entity. In a manufacturing model an entity may be a part, a subassembly, a finished product, a carton, a work order — anything that moves.

Entities often have *attributes*. An attribute is a local data item of which each entity has its own copy. For example a part entity might have a color attribute, or a work order entity might have a due date attribute.

3.2 Resources

Resources are constructs used to model the state of constrained system elements. On a highway, for example, physical lane space is a constrained system element. In a manufacturing facility, production machines and material handling equipment are constrained system elements.

Most tools have one or several "resource" constructs that are intended for the *direct* representation of constrained system elements, and other constructs (e.g., counters, switches, data variables) whose state can be related to the state of such elements. We take the broad view that any of these constructs may appropriately be called a "resource."

A wide range of physical things can be modeled as *either* entities or resources. A machine operator could be viewed as either or both, as could a forklift truck. In fact it is possible to invert the world view: consider a fixed-location grinding machine "entity" that goes through time looking for a constrained supply of a parts "resource" to process.

3.3 Simultaneous Movement of Entities

If a program written in a procedural language such as FORTRAN or C is thought of as a flow chart, there is one "entity" (one "unit of control") that circulates through the flow chart until the program has finished. In discrete event simulation software, there may be dozens or even thousands of individual entities that circulate simultaneously through the network. They stop and start and hand off control to one another in a way that is carefully choreographed inside the simulation software.

Despite this "multi-threaded" complexity, the typical underlying simulation program — which itself is a procedural program — can only recognize a single *active entity* at any moment during the computer's execution of the model. Broadly speaking the active entity will move through the flow chart until it encounters a *delay* (a waiting condition) or is destroyed. It will then yield to some other entity which becomes the new active entity. And so on. It is in this manner that simulation software handles simultaneous events serially.

3.4 Entity States

Each entity is always in one of five alternative states: the Active State; the Ready State; the Time-Delayed State; the Condition-Delayed State; the Dormant State. We designate these States 1 through 5, respectively, and capitalize the state names throughout the paper for emphasis. All simulation software recognizes all five entity states and handles each state in a particular way.

3.4.1 State 1: The Active State

The Active State is the state of the active entity. Many entities may move into and out of the Active State at any given simulated time. During the underlying execution of the EMP, there is only one active entity at any moment of computer time. In part/machine terms, a part is in the Active State while it is actually moving through the flowchart (executing logic or calculations), but not while it is waiting for or using a machine.

3.4.2 State 2: The Ready State

The Ready State is the state of entities ready to enter the Active State. Usually more than one entity is processed during an instant of simulated time. The CUP usually produces at least one entity that is free to move through the network of operations, and EMP execution often produces other free-to-move entities that must all be processed during that EMP. Yet only one entity at a time can be in the Active State.

So, at a given point during an EMP, there may be zero to many entities that are free to move through the network but cannot do so for the sole reason that some other entity is active. These Ready-State entities are not waiting for anything other than their turn to move.

3.4.3 State 3: The Time-Delayed State

The Time-Delayed State is the state of entities waiting for a known simulation time to be reached so that they can then (re)enter the Ready State. Time-Delayed entities are waiting for a certain amount of time to elapse. A "part" entity might wait for example for the end of its ongoing processing time when it is in the midst of using a machine. For Time-Delayed entities, the "certain amount of time" may have been specified as a random variable or a state-dependent value, but it became known at the *beginning* of the wait, thereby making it possible for the entity to wait for a scheduled next time when it can re-enter the Ready State.

3.4.4 State 4: The Condition-Delayed State

The Condition-Delayed State is the state of entities waiting until some system condition is met — such as a part waiting to use a machine that is currently busy. Condition-Delayed entities are waiting for a specific condition whose time of occurrence cannot be determined at the beginning of the delay.

3.4.5 State 5: The Dormant State

Sometimes it is desirable to place entities into a state from which there is no escape that will be triggered *automatically* by changes in model conditions. We call this the Dormant State. Dormant-State entities rely on other entities to make an explicit decision to bring them from the Dormant State back into the Ready State.

An example is the placement of job-ticket entities into a waiting state (Dormant State) that requires a control entity or operator entity to make an explicit decision about which job-ticket to pull next.

4 ENTITY MANAGEMENT STRUCTURES

In order to understand how an Entity Movement Phase proceeds, one needs to understand the data structures that are used by simulation software to separate and organize entities in each of the five states.

An entity is itself a fairly simple data structure occupying a few dozen to a few hundred bytes (characters) of computer memory. The data that "is" each entity never moves in computer memory, not even for the active entity. Instead, the simulation software uses a variety of *ranked* (ordered) *lists* and other data structures to organize and track the entities.

4.1 The Active Entity

As already stated, there is only one active entity at a time. It can be thought of as occupying a list of length one. This Active-State entity flows through the network until encountering an operation that puts it into another state or destroys it. It then yields control to another entity. If there is no possibility of further action at the current time, the EMP ends and a CUP begins.

4.2 The Current Events List

Entities in the Ready State belong to a single ranked list that we'll call the *current events list*. This list has a tool-dependent name and is managed differently by each tool, so we will come back to it in the tool-specific sections.

For all three tools studied there are various ways an entity can join the current events list. Most commonly it is through migration from the future list or from a delay list (these lists are defined below). Also, new "cloned" entities start out on the current events list.

4.3 The Future Events List

Entities in the Time-Delayed State are inserted into a single *ranked list* at the beginning of their delay of known duration. This list, which we call the *future events list* (FEL), is ranked by increasing entity *move*

time. Entity move time is calculated as the simulation clock value at the time of entity insertion into the FEL, plus the known delay duration. The *event* at the head (front) of the FEL is the next event to occur after completion of an ongoing EMP.

After an EMP is over, the CUP looks at the FEL, and the move time of the FEL's earliest-ranked member becomes the new clock value. If this event is the scheduled (re)activation of one or more entities that the modeler has defined, then these entities are shifted to the Active State and/or the Ready State and the EMP begins.

Some tools will pull additional events from the front of the FEL during one CUP if those events have move times that match the move time of the head of the FEL.

In addition to an existing entity beginning a time delay, there are some other ways entities and/or *internal events* can get onto the FEL, depending on the tool. These include entity arrivals and beginning- and end-of-downtime events, as well as others.

4.4 Delay Lists

Delay lists are used to manage entities in the Condition-Delayed State. When two or more such entities wait for identical or partially identical conditions, competition results. There are different ways to implement competition. We describe two basic ways in this section and a third way in Section 4.5. Important software-specific aspects are described in later sections.

If the delay condition can be related to one or more specific changes in the state of the model, then *related waiting* can be used. For example, when a machine changes from busy to idle, the underlying algorithms can fetch the next entity to use the machine from the appropriate delay list. Related waiting is the most prevalent approach used to structure delay. It offers execution efficiency and precise selection of entities in pre-ranked delay lists. Entities undergoing related waiting are checked for possible removal from the Condition-Delayed State when the related model state changes.

If the delay condition is too complex to be related to model state changes, *polled waiting* may be useful. With polled waiting the underlying simulation algorithms assume responsibility for checking from time to time to see if the waiting entity(ies) can be removed from the Condition-Delayed State. Delay lists for polled waiting must be pre-ranked because the polling algorithm must make decisions about which entities to check first.

Complex delay conditions include Boolean (AND/OR) combinations of possible task-triggering state changes (e.g., a part supply running low prior to 2:00 PM *or* an output bin needing to be emptied).

4.5 Independent Lists

Entities in the Dormant State reside in special lists that are neither related to a delay condition nor polled. We call these *independent lists*. In general there is no automatic way for an entity to leave an independent list. Dormant entities are waiting for *something*, but they don't know what the something is: it's the job of some other entity to know. The "something" could involve resource constraints, which makes this the third of the methods mentioned above for managing competition. Independent lists are always defined by the modeler.

5 HOW COMPETITION IS MANAGED IN THREE SOFTWARE TOOLS

We have chosen three software tools for explicit description. The tools are SIMAN V (Systems Modeling Corporation, Sewickley, PA, USA), ProModel (Version 1.1 for Windows, ProModel Corporation, Orem, UT, USA), and GPSS/H (Release 2, Wolverine Software Corporation, Annandale, VA, USA). There are many other tools that might be as well or better suited for a particular task than the ones described here. Our choice has been made based on our perception that these three tools are fairly general-purpose, i.e., applicable in diverse contexts.

5.1 SIMAN

The discussion of SIMAN addresses SIMAN V, which has some features that differ in important ways from earlier versions of SIMAN.

Table 1 shows the SIMAN names for the constructs discussed in previous sections. The SIMAN documentation also mentions the *Event Calendar*, which is the combined Current Events Chain and Future Events Heap.

Generic Term	SIMAN Equivalent
Entity	Entity
Resource	Resource, Blockage, Conveyor, Transporter
Operation	Block
Current Events List	Current Events Chain
Future Events List	Future Events Heap
Delay List	Attached Queue
Independent List	Detached Queue

Table 1: SIMAN Terminology

5.1.1 The Current Events Chain

SIMAN has a Current Events Chain (CEC) that contains all Ready-State Entities (and only contains Ready-State Entities). The first step in the EMP is to remove the Entity at the head of the CEC and place it into the Active State. The Active-State Entity is not part of the CEC.

If additional Ready-State Entities are placed on the CEC while the Active-State Entity is moving, they are inserted in *last-in, first-out* order. This means that if one part splits into two, then after the original part comes to rest its clone will be the next active Entity. An exception is that if several Ready-State Entities are added

to the CEC simultaneously from the same operation, they will be added at the front of the list (that is, LIFO in terms of other Ready-State Entities) but they will be FIFO among themselves.

When the most recently active Entity leaves the Active State and there are no more Ready-State Entities, (i.e., the CEC is empty), the EMP checks for polled wait conditions that might have been relaxed (see Section 5.1.3.2). If there are none a CUP ensues.

5.1.2 The Future Events Heap

Time-Delayed Entities in SIMAN reside in an internal structure named the Future Events Heap. From the modeler's point of view this structure behaves like a list ranked on increasing move time. The Entity with the earliest move time is the next one off the FEH.

SIMAN will remove more than one Entity from the FEH during a single CUP if the Entities are tied for earliest move time.

The SIMAN FEH may contain "internal Entities" that come not from other Entity states but from elements specified in the model/experiment definition. An example is beginning- and end-of-downtime Entities. These are not really "Entities" but are system events. When such an Entity is found during a CUP, appropriate processing ensues and zero or more "real" Entities may end up in the Ready State (with the leader eligible for Active-State status immediately). Because of internal Entities, the CEC may be empty when an EMP begins. The check for polled wait conditions (see Section 5.1.3.1) will nevertheless be performed as part of the EMP.

5.1.3 Queues

SIMAN has several types of delay lists. Those directly accessible to the modeler include Attached Queues and Detached Queues, as discussed in the following sections.

5.1.3.1 Attached Queues

Attached Queues are Entity lists used by SIMAN to implement the Condition-Delayed State. "Attached" Queues are attached to a Block — a particular Block of a type known as a Hold Block. For example, SEIZE, the Block used by an Entity (e.g., a part) to attempt to capture a Resource (e.g., a machine), is a Hold Block. Every Hold Block has a Queue attached to it in which Entities can wait for the Hold condition to be satisfied.

Attached Queues are ranked FIFO, LIFO, or by lowest or highest value of an expression at insertion time. The Entities in a given Attached Queue might be waiting for different conditions, e.g., different Resources or combinations of Resources.

Waiting in Attached Queues is related to the underlying condition (except in the case of the SCAN Block; see the last paragraph in section 5.1.3.1). When the Hold Block condition changes, the "winning" Entity is chosen from among a list of Contenders. The Contenders consist of the highest-ranked Entity waiting for that condition (or waiting in part for that condition) in each of one or more Queues. If only a single Queue is involved, then there is only one Contender.

Selection of the winning Entity from among multiple Contenders, if necessary, is made primarily by Block-based Priority and secondarily by FIFO (see Section 6.8).

Other Hold Blocks in SIMAN that implement related waiting include ACCESS, ALLOCATE, PREEMPT, PROCEED, REQUEST, and WAIT.

A Shared Queue is like an Attached Queue but can be referenced in more than one place. A Shared Queue allows Entities waiting at different Hold Blocks to be ranked in one list. For a given condition there can be at most one Contender in a Shared Queue.

If no QUEUE Block precedes a Hold Block, SIMAN will generate an attached, non-shared Queue called an Internal Queue. Internal Queues are similar to Attached Queues but are not named, are always ranked FIFO, produce fewer statistics, and are more execution-efficient.

Special cases of Hold Blocks are SCAN and WAIT. SCAN holds Entities until an arbitrary expression that can reference system state information and/or data values becomes true. SCAN Queues are polled. WAIT holds an Entity until a signal code is received. Because a signal can only be sent by another Entity, WAIT is an implementation of the related wait discussed above.

5.1.3.2 Detached Queues

Detached Queues are Entity lists used by SIMAN to implement the Dormant State. Entities in Detached Queues can be "sprung" from their Dormant State by SEARCH/REMOVE Block pairs.

Some useful options for ranking Detached Queues on insertion and for re-ranking and choosing on extraction are provided (see Section 6.9).

Entities in Detached Queues can also be extracted when QPICK or MATCH Blocks execute.

5.2 ProModel

Table 2 shows the ProModel names for the constructs discussed in previous sections. Although the terminolo-

Generic Term	ProModel Equivalent
Entity	Entity
Resource	Location, Resource, Variable, Node
Operation	Process Step
Current Events List	Action List
Future Events List	Future Events List
Delay List	Waiting List
Independent List	None (but see Section 5.2.3)

Table 2: ProModel Terminology

gies may differ, the mechanisms described in this section also apply to the MedModel and ServiceModel products from the same vendor.

ProModel Entities compete for *Locations, Resources* and *Variables.* A Location corresponds to the physical space occupied by an Entity. An Entity can occupy no more than one Location at any given computer time during the execution of the model. This means an Entity must gain access to its next Location before it can let go of its current one

ProModel Resources are used to model resources that are auxiliary to the physical space required. Examples are forklift trucks or human beings. Entities can own and control multiple Resources simultaneously.

Resources themselves can compete for *Nodes*, moving independently through the Node network in search of something to pick up or a place to be idle. In this sense Resources can behave internally like Entities. They migrate among the first four of the Entity states and are tracked in lists.

A Variable in ProModel is a general-purpose data storage element whose state can be the object of a WAIT UNTIL and for which statistics are automatically collected.

5.2.1 The Action List

The ProModel Action List contains Entities (and Resources) in the Ready State. It is ranked LIFO and is empty at the end of the EMP. Deactivation of the active Entity or Resource causes the first Entity or Resource on the Action List to become active.

5.2.2 The Future Events List

Entities (undergoing WAIT operations) and Resources (while moving), along with certain internally generated Entities and events, can wait on the Future Events List (FEL). Processing is "first out based on earliest move time." ProModel will remove only one Entity or event during a CUP. In the case of time ties on the FEL there can thus be two or more successive EMPs that use the same instant of simulated time.

Many of the ProModel model-definition constructs have optional user-defined *Logic* fields (for example, Downtime Logic and Location Exit Logic). Logic is a collection of Operation Statements that are automatically executed at the appropriate point during model execution. An Entity can launch *Independent Logic* which is like a cloned subroutine call (the Entity goes on its way). We mention Logic here because Downtime Logic and Independent Logic can produce non-Entity-related events on the FEL. When processed, these events may go into another future-list wait or into some type of delay list. They may cause Entities (or Resources) to materialize on the Action List.

5.2.3 Waiting Lists

ProModel's *Waiting Lists* function as delay lists to implement a variety of related waiting conditions. There is no polled waiting. One type of Waiting List (Entities waiting for a SEND) is more like an Independent List.

To understand the interaction among Locations, Resources, and Variables, we need to take a peek inside the structure of ProModel. The transaction-flow part of ProModel is specified by the modeler as an ordered collection of *Process Steps* called the *Process Table.* Every Process Step contains the name of an Entity Type (or *All*) and the name of a Location (or *All*). An Entity "flows" from one Process Step to the next by jumping to the next Process Step in the Table that matches its Type and Location (starting over again at the top of the Table if necessary). This determines "what this Entity Type is supposed to do when it is at this Location."

A Process Step has two logical components, *Operation Logic* and *Routing Logic,* and can contain zero, one, or more than one of each component. Competition by Entities for Resources takes place in the Operation Logic. Competition by Entities for Locations and any transportation Resources takes place in the Routing Logic. The Routing Logic is applied after the Operation Logic has been executed.

A Waiting List (for Entities) is attached to each Location, to each Resource, and to each Variable. A Waiting List (for Resources) is attached to each Node. (Competition by Resources for Nodes takes place automatically based on Path Networks, Work/Park Lists, and Node/Location associations defined outside the table of Process Steps.)

A single Entity (or Resource) can reside simultaneously in many delay lists of the same type. As a result, ProModel does not require a polling mechanism for modeling Boolean (AND/OR) combinations of conditions. An internal mechanism removes the Entity from the "other" delay lists as soon as it escapes from one of them. For more on Boolean conditions in ProModel see Section 6.5.

There are various Routing Rule options for specifying next Location alternatives. And it is possible to define a Location in such a way that it can override the ranking of its delay list when it is ready to accept another occupant.

ProModel has no independent lists as such. However, JOIN, LOAD, and SEND are all Routing Logic options that place Entities on special Location-specific lists where they await a JOIN, LOAD, or SEND Operation Statement, respectively, executed by another Entity at the destination Location. This explicit triggering makes these special lists *resemble* independent lists. But because of the Location relationship, and because the condition is somewhat specific, and finally because the lists are not custom-managed, we consider the waiting to be related waiting and the lists to be delay lists.

5.3 GPSS/H

GPSS/H equivalents of the generic terms are given in Table 3. Note that the Current Events Chain serves two purposes as described below.

The EMP in GPSS/H is called the *Scan Phase*. The GPSS/H Scan Phase is more involved than the EMP in SIMAN and ProModel. (GPSS/H Scan Phase particulars are discussed in chapters 4 and 7 of Schriber 1991.)

Generic Term	GPSS/H Equivalent
Entity	Transaction
Resource	Facility, Storage, Logic Switch
Operation	Block
Current Events List	Current Events Chain
Future Events List	Future Events Chain
Delay List	Current Events Chain
Independent List	User Chain

Table 3: GPSS/H Terminology

5.3.1 The Current Events Chain

Perhaps the most striking difference in GPSS/H when compared with other tools is that certain Condition-Delayed Transactions are commingled with the Ready-State Transactions on the Current Events Chain (CEC). For Condition-Delayed CEC Transactions, the CEC can be thought of as a single global delay list.

Other than the CEC and some internal delay lists (see Section 6.6), there are no delay lists in GPSS/H. (GPSS/H has a QUEUE Block and a Queue construct that do not perform list management functions; they are for statistics gathering purposes only.)

Another unique characteristic of the GPSS/H CEC is that it is ranked FIFO within Priority Class. See Section 6.8 for more on Priority. This reflects its global-delay-list function. Like other types of delay lists, the CEC is frequently *not* empty in GPSS/H when the EMP ends.

5.3.1.1 The Scan Phase

At the beginning of the Scan Phase (EMP) GPSS/H starts at the head of the CEC and tries to move that Transaction into its next Block. If the Block is one that can deny entry (SEIZE, ENTER, GATE, TEST or PREEMPT) and entry is denied, then the Transaction is in a Condition-Delayed State and GPSS/H *leaves the candidate on the CEC* and moves on to examine the sequential Transaction on the CEC. If entry is not denied, then the candidate becomes the active Transaction (without being removed from the CEC) and begins executing Blocks.

The same mechanism applies whenever a Transaction becomes active. If entry is denied when the active Transaction tries to execute a Block, then the Transaction shifts to the Condition-Delayed State and remains on the CEC while GPSS/H resumes scanning the CEC in search of a new active Transaction. However, because of possible state changes precipitated by the active Transaction's Block execution(s), the scan can either continue sequentially or *restart* (see Section 5.3.1.2 below).

The GPSS/H mechanism of keeping certain Condition-Delayed Transactions on the CEC and examining them one or more times during the Scan Phase to see if they are in Ready State at the instant of examination implies that all of these Transactions are fundamentally in a polled wait condition.

5.3.1.2 Restarting the Scan

There is an internal status change flag (SCF) that is set to TRUE when any of certain *unique blocking* conditions (see Section 5.3.1.3) changes state. If the SCF is TRUE when the active Transaction ceases to be active, then the SCF is set back to FALSE and the scan restarts at the head of the CEC as if the EMP had just begun; otherwise the scan continues sequentially on the CEC.

The rationale behind this approach is that there may be up-stream Transactions that have higher Priority or at least arrived sooner and should be given first crack at moving in response to the change in system state. The net effect of scan restarts is to provide FIFO-within-Priority-Class queueing automatically for all operations involving the most common blocking conditions.

It is possible for the Scan Phase to end (i.e., the scan reaches the CEC tail and the SCF is FALSE) with Ready-State Transactions still on the CEC. Such "missed" Ready-State Transactions might have been waiting for a non-unique condition that became true. In these rare cases, a BUFFER Block can be executed by the active Transaction to return itself temporarily to the Ready State and force an immediate scan restart.

5.3.1.3 Related Waiting on the CEC

State changes involving unique blocking conditions include (but are not limited to) the transition of a resource (Facility) into or out of use; the transition of a Storage (a GPSS/H counter with a capacity) to a smaller count, or out of the empty or into the full state; and a change in the setting of a true-or-false Logic Switch. Transactions waiting to SEIZE a Facility or ENTER a Storage or waiting at a GATE for a Storage to become non-empty or full or for a Logic Switch to change are in a unique blocking condition. (Other types of unique blocking are also possible but are not detailed here.)

Scan restarts imply extra processing demands while GPSS/H re-encounters and re-evaluates Condition-Delayed Transactions. To combat this each Transaction has a flag called the Scan Skip Indicator (SSI) that marks certain Transactions — those waiting for unique blocking conditions — as Condition-Delayed. This flag is checked before an actual attempt is made to move a candidate-for-active Transaction into its next Block, allowing the scan to bypass quickly most Condition-Delayed Transactions.

The SSI gets cleared automatically at the instant the unique blocking condition for which the Transaction is

waiting gets removed. Internal delay lists are used to track which Transactions' SSIs need to be cleared for a given state change. These lists *are* related to the underlying condition, so the fundamental polled waiting nature of the GPSS/H CEC mechanism is in fact — for unique blocking conditions — a hybrid polled/related approach for unique blocking conditions. (It is primarily polled but is supported, for execution efficiency, by a related-list mechanism.)

5.3.2 The Future Events Chain

The GPSS/H Future Events Chain (FEC) is like future events lists in other tools. The GPSS/H CUP will remove multiple Transactions from the FEC if they are tied for the earliest move time, inserting them one by one into the appropriate place on the CEC.

GPSS/H does not schedule internal entities for beginning- and end-of-downtime events. GPSS/H uses model downtimes (as well as many other control conditions) with actual Transactions. These are ordinary Transactions that go through the ordinary Time-Delayed State to simulate time-between-failures and time-to-repair.

5.3.3 User Chains

Transactions are put into a Dormant State in GPSS/H via the User Chain construct. User Chains are independent lists. After a Transaction puts itself onto a User Chain (via a LINK Block), it can only be removed by another Transaction (via an UNLINK Block). When UNLINK execution transfers one or more Dormant-State Transactions to Ready State, the SCF will be made TRUE so that these CEC newcomers will have their turn to become active before the next CUP. User Chains can achieve performance improvements over CEC-based queueing because User Chains (like delay lists in other tools) need never be scanned except when an UNLINK is executed.

6 WHY IT MATTERS

6.1 Overview

In this section we list several types of modeling situations and comment on them in terms of the three software tools. The situations are stated in a generic textbook way, but real world analogues are included. Many simulation projects can be carried out without encountering these situations. However, the situations do provide a framework for exposing specific differences among the internal algorithms of the three tools studied.

6.2 Yielding Control

Suppose the arrival of the last carton is to trigger the releasing of a cluster (slug) of cartons from a conveyor. The last carton is to "close the gate" against the arrival of more cartons, but it must let waiting cluster-mates get through the gate first. Or, in another example, a box-of-parts entity is to split (clone) individual parts into the system and the parts need first crack at some resource.

Of interest is whether a mechanism is available to allow the Active-State entity to yield control to newly created clones or to other entities that have shifted into the Ready State.

In SIMAN, clones created via BRANCH can be allowed to go ahead of the original Entity. An alternative to BRANCH is to use a DELAY to put the active Entity into a Time-Delayed State for a brief simulated time.

In ProModel, "WAIT 0" can be used to put the active Entity back on the FEL. It will be returned later to the Active State at the same simulated time. One of ProModel's cloning operations, CREATE, can be used to allow the clones to go ahead of the cloner.

In GPSS/H, "PRIORITY PR,BUFFER" can be used to reposition the active Entity behind equal-priority Transactions (including any clones) on the CEC, shift the active Entity back to the Ready State, and restart the scan of the CEC.

6.3 Re-capturing the Same Resource

Suppose in a model a part relinquishes a machine, then immediately re-competes for the machine (e.g., RELEASE followed by SEIZE in GPSS/H or SIMAN; or FREE followed by GET or USE followed by USE in ProModel). The intention is to give a more highly qualified part a chance to be the next to capture the machine.

Of interest in this scenario is the order of events following the relinquishing of a resource. There are at least three logical alternatives: (1) Coupled with the relinquishing of the resource is the immediate choosing of the next owner of the resource, without the relinquishing entity having yet reached the point of becoming a contender. (2) The step of choosing the next resource owner is deferred until after the relinquishing entity has become a ranked contender. (3) "Neither of the above" — that is, choice of the next owner is not coupled with the relinquishing of the resource, but the active entity does not contend with others waiting for the resource, either; instead, without paying heed to others, it recaptures the resource immediately.

Each of these alternatives comes into play in the tools considered here. SIMAN, ProModel, and GPSS/H implement the first, second and third alternatives, respectively. (We are talking about default behavior that can be modified using other techniques.)

6.4 List Head's Request Can't Be Satisfied but Another's Can

Suppose several Condition-Delayed entities are waiting in FIFO order in a single place for at least one unit of a particular resource. The second entity wants only one

unit of that resource. The first entity wants two units or one unit plus some other condition. If the first request cannot be satisfied but the second one can, will the second entity get the resource?

In SIMAN, the second Entity will not get the Resource because only the highest-ranked member of the Queue is a Contender (see Section 5.1.3.1). Other techniques can be used to modify this behavior.

For ProModel Resources the Waiting List is searched from the top. JOINTLY GET requests that are not satisfied are passed by. GET requests will claim as much of what is requested as possible, even if the request is not fully satisfied. But the first Entity in a Location's delay list always gets the Location, because an Entity can be requesting no more than one unit and Boolean Location conditions are always OR.

In GPSS/H with CEC queueing, the second Transaction will get the resource (if the scan reaches it before conditions change again). User Chains can be used to modify this behavior.

6.5 Waiting for a Compound Condition

All three tools offer a way to wait for the truth of a Boolean expression that describes a complex model state, but the tools differ in implementation and flexibility.

SIMAN's "related wait" mechanism (Section 5.1.3.1) allows waiting for combinations of Resources by using Resource Sets (for OR conditions) and lists of Resources and/or Resource Sets (for AND conditions). If the "winning" Entity does not meet the full condition, then no Entity claims the Resource.

SIMAN also offers the SCAN mechanism (Section 5.1.3.1). SIMAN waits to evaluate all SCANs until the CEC is empty. Then it evaluates *each* SCAN condition once on behalf of the Entity at the head of that SCAN's Queue, extracting that Entity to the CEC if the condition is true. Then, if the CEC is non-empty, the EMP continues. When the CEC is again empty, each SCAN condition is again evaluated on behalf of the Entity at the head of that SCAN's Queue, and so on. Although the polled SCAN mechanism might miss a transitory state change, the delayed evaluation does minimize overhead.

In ProModel the Operation statements for claiming Resources (GET, JOINTLY GET, and USE) and for waiting on a Variable level (WAIT UNTIL) can model AND/OR Boolean conditions. Boolean Location selection is only on an OR basis because an Entity can claim only one Location. GET *et al.* can only look at Resources and WAIT UNTIL can only look at Variables, Array elements or Entity Attributes.

For Boolean conditions involving GET or USE, once any Resource request that is *part* of the condition has been satisfied (but with the overall condition still being false), then only that "branch" of the Boolean tree is evaluated further. For example, in the condition GET A OR (B AND C), once a unit of B or C has been claimed then A alone cannot satisfy the condition.

ProModel attaches a delay list to each Variable, so the wait is related. Transitory changes in a Variable's value will not be missed by entities in a WAIT UNTIL.

GPSS/H offers the refusal-mode TEST Block which can deny entry (thereby forcing FIFO-within-Priority-Class queueing on the CEC) based on virtually any condition. However, there can be performance penalties. Scan-skip indicators are not set for TEST conditions, so the full condition must be evaluated every time each such Transaction is encountered in an EMP. TEST is a polled wait, so there is a slight chance of missing a very transitory state change unless BUFFER is used in some cases.

6.6 Grouping, Batching, Combining, and Matching

All three tools support options for making many parts from a box of parts or an assembly out of subassemblies. All allow entities to be collapsed into a single entity or collected in one place prior to moving forward as individual entities and allow single entities in different parts of the model to pause and coordinate with one another. ProModel and SIMAN also allow entities to be grouped into a compound Entity that behaves for a time as a single Entity but can be ungrouped later.

In ProModel all entities that reach a given collection point are collected together — a natural representation of many manufacturing systems. SIMAN allows the formation of many collections at one point.

GPSS/H can form one collection at each of many points, relying on a built-in attribute (the Assembly Set number) which defines a "family" as "all Transactions that share a common ancestor through one or more previously executed SPLIT Blocks." In all three tools the underlying mechanism for these options may be thought of as relying on internal related delay lists.

6.7 Signals

Suppose entities need to wait for a notification from another part of the model. All three tools provide this capability. SIMAN offers the WAIT Block which waits for a *Signal*. When the Signal is sent one or more Entities in one or more Queues can move. ProModel offers the WAIT UNTIL capability, allowing related waiting using a Variable as a signal. A Variable delay list is always searched from head to tail when the Variable's state changes. GPSS/H has true/false Logic Switches that provide a signalling capability based on polled/related waiting.

6.8 Priorities

All three tools support the concept of Priority — a numerical value that either can or does influence the insertion-time ranking of waiting entities. In SIMAN and ProModel, the priority is attached to the Hold Block (SIMAN) or attached to the Operation Step (ProModel).

Entities in relative waits are ranked by default as "FIFO by priority" meaning Priority is the dominating factor. Priority is a transitory thing and only affects the ranking for a particular wait.

In GPSS/H Priority is a built-in but modifiable Transaction attribute used in ranking Transactions on the CEC (only). (Care should be taken when deprioritizing if an immediate effect is desired. Sometimes a BUFFER Block is called for to force the active Transaction to yield control temporarily to higher-Priority Transactions.)

6.9 Extraction-time Ranking

For models that perform resource scheduling, it can be critical to use current system state information to extract the best candidate from a waiting list. All the tools offer many options for ranking on *insertion* into a delay list, but we are looking for *extraction-time ranking*.

SIMAN delivers expression-based SEARCHing that includes references to candidate attributes and identifies the best single candidate.

ProModel allows extraction of a best candidate based on one attribute of the candidates but not on an arbitrary expression that incorporates candidate attributes.

GPSS/H allows extraction (via UNLINK) based on a modeler-specified Boolean expression used as a filter; the expression can directly reference attributes of candidate-Transactions in the Dormant State.

6.10 Interactive Model Verification

This section comments on how a detailed understanding of "how simulation software works" encourages and supports interactive probing of simulation model behavior.

In general, simulation models can be run interactively or in batch mode. Interactive runs are of use in checking out (verifying) model logic during model-building and in troubleshooting a model when execution errors occur. Batch mode is then typically used to make production runs with verified models.

Interactive runs put a magnifying glass on a simulation model while it executes. The modeler can follow the active entity step by step and display at will the current and future events lists and the delay and independent lists as well as other aspects of the state of the model. These activities yield valuable insights into model behavior for the modeler who knows the corresponding concepts. Without such knowledge, the modeler might not take full advantage of the interactive tools provided by the software, and might even entirely avoid using the tools.

7 SUMMARY

This paper discusses the major logical considerations that motivate the underlying platform which is the basis for much discrete-event simulation software. The discussion centers on alternative entity states, the use of lists to organize entities in the various states, and the use of algorithms to manage these lists and manipulate entities during a simulation. Practitioners who are knowledgeable about these considerations in general and understand their implications in terms of the simulation software they use should be in a position to build models faster and more creatively, use simulation tools (including model verification tools) more imaginatively, and be more confident that the models they build reflect system complexities precisely as intended.

ACKNOWLEDGEMENTS

Much of the information in this paper was derived from extended conversations with software vendor personnel. The authors gratefully acknowledge the time investments in this project that were generously provided by David T. Sturrock, Deborah A. Sadowski, C. Dennis Pegden and Vivek Bapat, all of Systems Modeling Corporation; Charles Harrell, Eric Du and Kerim Tumay, all of ProModel Corporation; and Robert C. Crain and James O. Henriksen, both of Wolverine Software Corporation.

REFERENCES

Banks, J., B. Burnette, H. Kozloski, and J. Rose. 1994 (forthcoming). *Introduction to SIMAN V and Cinema V.* New York: John Wiley & Sons.

Henriksen, J. O., and R. C. Crain. 1989. *GPSS/H Reference Manual,* Third Edition. Annandale, Virginia: Wolverine Software Corporation.

Pegden, C. D., R. E. Shannon, and R. P. Sadowski. 1990. *Introduction to Simulation Using SIMAN.* New York: McGraw-Hill Inc.

ProModel Corporation. 1993. *ProModel for Windows.* Orem, Utah: ProModel Corporation.

Schriber, T. J. 1991. *An Introduction to Simulation Using GPSS/H.* New York: John Wiley & Sons.

Systems Modeling Corporation. 1994. *SIMAN V Reference Guide.* Sewickley, Pennsylvania: SMC.

AUTHOR BIOGRAPHIES

DANIEL T. BRUNNER founded System Flow, an independent simulation services firm, in March 1993. Prior to that he was with Wolverine Software Corporation where he managed simulation services, technical support, training, and product marketing activities. He holds a B.S. in Electrical Engineering from Purdue University and an MBA from The University of Michigan. Mr. Brunner served as Business Chair of the 1992 Winter Simulation Conference and is General Chair of the 1996 WSC. He is a member of IIE and SCS.

THOMAS J. SCHRIBER is a Professor of Computer and Information Systems at The University of Michigan. He teaches in a variety of areas while doing research and consulting in discrete-event simulation. He is a member of DSI, ORSA, SCS and TIMS.