Proceedings of the 1996 Winter Simulation Conference
ed. J. M. Charnes, D. J. Morrice, D. T. Brunner, and J. J. Swain

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

Systemflow Simulations, Inc.
6366 Guilford Avenue, Suite 310
Indianapolis, Indiana 46220-1750, U.S.A.

ABSTRACT

This paper provides simulation practitioners and interested simulation consumers with a grounding in how discrete-event simulation software works. Topics include discrete-event systems and modeling; entities, resources and operations; simulation runs; entity states; entity lists; and entity-list management. The implementation of these generic ideas in SIMAN, ProModel, and GPSS/H is described. The paper concludes with several examples of "why it matters" for modelers to know in fine detail how their simulation software works. (This paper is an updated version of an identically named paper appearing in the *Proceedings of the 1995 Winter Simulation Conference,* pp. 110-117.)

1 INTRODUCTION

1.1 Background

A "black box" approach is often taken in teaching and learning discrete-event simulation software. The external characteristics of the software are studied, but the foundation on which the software is based is ignored or the foundation is touched on only briefly. Choices made in implementation of the foundation might not be studied at all and related to step-by-step model execution. The modeler therefore might not be able to reason things through when faced with such needs as developing good approaches for modeling complex situations, using interactive tools to come to a rapid understanding of error conditions arising during model development, and using interactive tools to verify that complex system logic has been accurately captured in a model. The objective of this paper, then, is to present the logical underpinnings of discrete-event simulation and illustrate this material in terms of three implementations of discrete-event simulation software.

1.2 Structure of the Paper

In Sections 2, 3 and 4 we comment on the nature of discrete-event simulation; entities, resources and operations; and simulation experiments. Sections 5 and 6 deal with entity states and entity management structures. Section 7 discusses the implementation of the preceding generic material in terms of SIMAN, ProModel, and GPSS/H. Section 8 explores "why it matters."

1.3 Terminology and Conventions

Throughout this paper we use terms that we define as well as terms reserved by the developers of particular simulation tools. Terms we define are **boldfaced** on first use. Tool-specific terms are Capitalized or, where appropriate, are spelled out in ALL CAPS.

2 ABOUT DISCRETE-EVENT SIMULATION

2.1 The Transaction-Flow World View

The "transaction-flow world view" often provides the basis for discrete-event simulation. In this view, a system consists of discrete units of traffic that move ("flow") from point to point in the system while competing with each other for the use of scarce resources. The units of traffic are sometimes called "transactions," giving rise to the phrase "transaction flow."

Numerous systems fit the preceding description. Included are many manufacturing, health care, transportation, civil, communication, defense and information processing systems, and queuing systems in general.

2.2 The Nature of Discrete-Event Simulation

A discrete-event simulation is one in which the state of a model changes at only a discrete, but possibly random, set of time points, known as event times. Two or more traffic units often have to be manipulated at one and the same time point. Such "simultaneous" movement of traffic at a time point is achieved by manipulating units of traffic *serially* at that time point. This often leads to logical complexities in discrete-event simulation.

2.3 Discrete-Event Modeling Languages

The challenges faced by a *modeler* escalate for the *designer* of a modeling language. The designer must

take the logical requirements of discrete-event simulation into account in a generalized way. Choices and tradeoffs exist. As a result, although discrete-event simulation languages are similar in broad terms, they can and typically do differ in subtle but important particulars.

3 ENTITIES, RESOURCES AND OPERATIONS

The generic term **entity** is used here to designate a unit of traffic (a "transaction"). Entities instigate **events**. An event is a happening that changes the state of a model (or system). In a model of an order-filling system, for example, the arrival of an order, which is an event, might be simulated by bringing an entity into the model.

It is useful to distinguish between two possible types of entities, here referred to as **external entities** and **internal entities**. External entities are those whose creation and movement is explicitly arranged for by the modeler. In contrast, internal entities are created and manipulated implicitly by the simulation software itself. For example, internal entities might be used in some languages to simulate machine *failures*, whereas external entities might be used to simulate the *use* of machines by work-in-process.

The generic term **resource** is used to designate constructs that model system elements (such as servers) that can cause unwanted delays for units of traffic. All tools have "resource" constructs for the *direct* representation of servers, and other constructs (e.g., switches, counters, variables) whose state or value can be the basis for entity delay. The view here is that any of these constructs can at least sometimes appropriately be called a "resource."

An **operation** is a step carried out by or on an entity while it moves through a system. The operations applicable to a ship at a harbor might be these: arrive; capture a berth; capture a tugboat; get pulled into the berth; free the tugboat; load cargo; etc. "Capture" is an example of a compound operation, e.g., wait (if necessary) for a tugboat, then take control of the tugboat. Some operations require time to pass, e.g., the loading of cargo.

4 OVERVIEW OF MODEL EXECUTION

4.1 Experiments, Replications, and Runs

A simulation project is composed of **experiments**. Experiments are differentiated from each other by the use of alternatives in a model's logic and/or data. An alternate part sequencing rule might be tried, for example, or the quantity of various machines might be varied.

Each experiment consists of one or more **replications** (trials). A replication is a simulation that uses the experiment's 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.

A replication involves initializing the model, running it until a run-ending condition is met, and reporting results. This "running it" phase is called a **run**.

4.2 Inside a Run

During a run the simulation **clock** (an internally managed, stored data value) tracks the passage of *simulated* time (as distinct from *wall-clock* time). The clock advances in discrete steps (usually of unequal size) during the run. After all possible actions have been taken at a given simulated time, the simulation clock is advanced to the time of the next earliest scheduled event. Then the appropriate actions are carried out at this new simulated time, etc.

The execution of a run thus takes the form of a two-phase loop: "carry out all possible actions at the current simulated time," followed by "advance the simulated clock," repeated over and over again until a run-ending condition comes about. The two phases are here respectively called the **Entity Movement Phase** (EMP) and the **Clock Update Phase** (CUP).

5 ENTITY STATES

Entities migrate from state to state while they work their way through a model. An entity is always in exactly one of five alternative states, as detailed below.

5.1 The Active State

The **Active State** is the state of the currently moving entity. Only one entity moves at any instant of wall-clock time. This entity progresses through its operations nonstop until it encounters a delay. It then migrates to an alternative state (or leaves the system). Some other entity then becomes the next active entity. And so on.

5.2 The Ready State

During an Entity Movement Phase there may be *more than one* entity ready to move, and yet entities can only move (be in the Active State) one-by-one while wall-clock times goes by. The **Ready State** is the state of those entities waiting their turn to enter the Active State during the current Entity Movement Phase.

5.3 The Time-Delayed State

The **Time-Delayed State** is the state of entities waiting for a *known* future simulated time to be reached so that they can then (re)enter the Ready State. A "part" entity is in a Time-Delayed State, for example, while waiting for the future simulated time at which an operation currently being performed on it by a machine will come to an end.

5.4 The Condition-Delayed State

The **Condition-Delayed State** is the state of entities delayed until some specified condition comes about, e.g., a "part" entity might wait in the Condition-Delayed State

until its turn comes to use a machine. Condition-Delayed entities are removed *automatically* from the Condition-Delayed state when conditions permit.

5.5 The Dormant State

Sometimes it is desirable to put entities into a state from which no escape will be triggered automatically by changes in model conditions. We call this state the **Dormant State.** Dormant-State entities rely on modeler-specified logic to transfer them from the Dormant State back to the Ready State. Job-ticket entities might be put into a Dormant State, for example, until an operator entity decides which job-ticket to pull next.

6 ENTITY MANAGEMENT STRUCTURES

Simulation software uses the following lists to organize and track entities in the five entity states.

6.1 The Active Entity

The active entity forms a single list of length one. This "list" is not given a name here. The Active-State entity moves nonstop until encountering an operation that puts it into another state (transfers it to another list) or removes it from the model. A Ready-State entity then becomes the next Active-State entity. Eventually there is no possibility of further action at the current time. The EMP then ends and a Clock Update Phase begins.

6.2 The Current Events List

Entities in the Ready State are kept in a single list here called the **current events list** (CEL). Entities migrate to the current events list from the future events list, delay lists, and user-managed lists. (Each of these latter lists is described below). In addition, any entities cloned from the Active-State entity usually start their existence on the current events list.

6.3 The Future Events List

Entities in the Time-Delayed State belong to a single list into which they are inserted at the beginning of their time-based delay. This list, called the **future events list** (FEL) here, is usually ranked by increasing entity **move time.** (Move time is the simulated time at which an entity is scheduled to try to move again.) At the time of entity insertion into the FEL, the entity's move time is calculated by adding the value of the simulation clock to the known (sampled) duration of the time-based delay.

After an Entity Movement Phase is over, the Clock Update Phase sets the clock's value to the move time of the FEL's highest ranked (smallest move time) entity. This entity is then transferred from the FEL to the CEL, migrating from the Time-Delayed State to the Ready State and setting the stage for the next EMP to begin.

The preceding statement assumes there are not *other* entities on the FEL whose move time matches the clock's updated value. In the case of move-time ties, some tools will transfer all the time-tied entities from the FEL to the CEL during a single CUP, whereas other tools take a "one entity transfer per CUP" approach.

Languages that work with internal entities usually use the FEL to support the timing requirements of these entities. The FEL is typically composed both of external and internal entities in such languages.

6.4 Delay Lists

Delay lists are lists of entities in the Condition-Delayed State. These entities are waiting for a condition to come about (e.g., waiting their turn to use a machine) so they can be transferred automatically into Ready State on the current events list. Delay lists, which are created automatically by the simulation software, are managed by using **related waiting** or **polled waiting**. (The approach taken may depend on the particular software being used.)

If delay can be related easily to events in the model that eliminate the delay, then related waiting can be used to manage the delay list. For example, suppose a machine's status changes from busy to idle. In response, the software can then automatically remove the next machine-using entity from the appropriate delay list and put it in Ready State on the current events list. Related waiting is the prevalent approach used to manage delay.

If the delay condition is too complex to be related easily to delay-resolving events in the model, polled waiting can be used to manage the delay list. With polled waiting the software checks regularly and routinely to see if entities can now be transferred automatically from delay lists to the Ready State.

Complex delay conditions for which polled waiting can be useful can include Boolean (*and/or*) combinations of state changes, e.g., a part supply runs low *or* an output bin needs to be emptied.

6.5 User-Managed Lists

User-managed lists are lists of entities in the Dormant State. The modeler must take steps to establish such lists and provide the logic needed to transfer entities to and from the lists. (The underlying software has no way to know why entities are put into user-managed lists and so has no basis for removing entities from such lists automatically.)

7 IMPLEMENTATION IN THREE TOOLS

The three tools chosen here for commentary on implementation particulars are Systems Modeling Corporation's SIMAN V; ProModel Corporation's ProModel, Version 3; and Wolverine Software Corporation's GPSS/H, Releases 2.x and 3. (See the References.) These three are among more than *fifty* tools

reported in 1995 for discrete-event simulation (Swain 1995). Some other tools might be better suited than any of these three for particular modeling activities, but we think that these three tools are representative.

7.1 SIMAN

SIMAN V equivalents of the generic terms in earlier sections are given in Table 1. For example, SIMAN uses *Blocks* to specify operations for *Entities*.

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 or Internal Queue
User-Managed List	Detached Queue

Table 1: SIMAN Terminology

7.1.1 The Current Events Chain

The current events list is called the Current Events Chain (CEC) in SIMAN. The SIMAN CEC is composed of all Ready-State Entities. The first step in SIMAN's Entity Movement Phase is to remove the Entity from the head of the CEC and make it active.

If Entities are placed on the CEC while the Active-State Entity is moving, they are inserted in *last-in, first-out* order. For example, if an Entity produces a **clone** (a copy of itself placed immediately on the CEC), then after the clone-producing Entity is no longer active, its clone will be the next Active-State Entity. If the Active-State Entity produces two or more clones simultaneously, they will be inserted at the front of the CEC (that is, LIFO in terms of other Ready-State Entities) *but* they will be FIFO among themselves.

When the active Entity leaves the Active State and there are no more Ready-State Entities, the EMP checks all polled wait conditions, transfers any qualifying Condition-Delayed Entities to the Ready State on the CEC, etc., until no more action can be taken at the current simulated time. Then the next CUP takes place.

7.1.2 The Future Events Heap

Time-Delayed Entities in SIMAN reside in a structure named the Future Events Heap (FEH). This structure behaves like a list ranked on increasing move time. The Entity with the earliest move time is the next one off the Future Events Heap when a Clock Update Phase occurs.

If there are ties for earliest move time, SIMAN will remove all the tied Entities from the Future Events Heap during one and the same Clock Update Phase.

The FEH can contain internal Entities resulting from elements specified by the modeler in the SIMAN experiment file. An example is beginning-of-downtime and end-of-downtime Entities. When an internal Entity is encountered on the FEH during a CUP, it is processed immediately. (In contrast, external Entities are simply put into Ready State on the CEC.) Because of internal Entities, the CEC might be empty after a CUP takes place. An EMP nevertheless takes place, insuring that a timely check of polled wait conditions will be made.

7.1.3 Attached and Internal Queues

Attached and Internal Queues are the two types of SIMAN lists containing Entities engaged in related waiting. Related waiting results from the use of Hold Blocks. For example, SEIZE, the Block used by an Entity (e.g., a part) to capture a Resource (e.g., a machine), is a Hold Block. Other related-waiting Hold Blocks are ACCESS, ALLOCATE, PREEMPT, PROCEED, REQUEST, and WAIT.

Associated with each Hold Block is an Attached or Internal Queue in which Entities wait for the Hold condition to be satisfied. If a QUEUE Block immediately precedes a Hold Block, an Attached Queue results. An Attached Queue is a *named* list of Entities waiting to execute the associated Hold Block.

Sometimes it is convenient to use identical Hold Blocks at multiple points in a model, e.g., to use a "SEIZE DRILL" Block two or more places in a model. The modeler can choose to associate with each Hold Block its own Attached Queue. These are *unshared* Attached Queues, because *two or more* lists of Entities then wait for the *same* Resource. Hold-Block Priority is used to determine the next Entity to get the Resource.

Alternatively, the modeler can put Entities delayed at two or more identical Hold Blocks into *one and the same* Attached Queue, a *shared* Attached Queue. Shared Attached Queues require use of the keyword SHARED in the QUEUES element in the SIMAN experiment file.

Entities are put into Attached Queues FIFO or LIFO, or are inserted into the Queues based on the value of a modeler-supplied expression.

If no QUEUE Block precedes a Hold Block, SIMAN provides an Internal Queue for that Hold. An Internal Queue is an *unnamed, non-sharable FIFO* Queue.

The SCAN Hold Block is an exception to the rule that Hold Blocks implement related waiting. A SCAN Block delays Entities until a user-supplied expression (involving system-state information and/or data values) is true. Queues (delay lists) that form at SCAN Blocks are polled at the end of each Entity Movement Phase.

7.1.4 Detached Queues

Detached Queues are Entity lists used by SIMAN to implement the Dormant State. Entities are put into Detached Queues when they execute QUEUE Blocks at

which a DETACHED modifier is specified. Such Entities are later transferred from their Dormant State to the Ready State by other Entities that use SEARCH and REMOVE or QPICK and MATCH Blocks for this purpose.

7.2 ProModel

ProModel equivalents of the earlier generic terms are given in Table 2. In ProModel, *Entities* compete for *Locations* and *Resources* and engage in operations on the basis of *Operation* and *Routing Statements*.

Generic Term	ProModel Equivalent
Entity	Entity
Resource	Location; Resource; Variable; Node
Operation	Operation Statement; Routing Statement
Current Events List	Action List
Future Events List	Future Events List
Delay List	Waiting List
User-Managed List	None (but see 7.2.3)

Table 2: ProModel Terminology

A Location models the physical space an Entity occupies. An Entity can only occupy one Location at a time. ProModel Resources are used to model resources other than Locations, e.g., forklift trucks; humans. Entities can control multiple Resources simultaneously.

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

A ProModel *Variable* is a general-purpose data element whose value can be the object of a WAIT UNTIL and for which ProModel collects statistics.

7.2.1 The Action List

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

7.2.2 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 per CUP. In case of time ties there will be successive EMPs that occur at the same instant of simulated time.

Many ProModel model-definition constructs have optional user-defined *Logic* fields, e.g., Downtime Logic and Location Exit Logic. Logic is a collection of Operation Statements automatically executed when appropriate. An Entity can launch *Independent Logic* which is like a subroutine call that is to be executed by a clone. 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-Events-List wait or into some type of delay list. They may cause Entities (or Resources) to materialize on the Action List.

7.2.3 Waiting Lists

ProModel's *Waiting Lists* function as delay lists to implement a variety of related waiting conditions. There is no polled waiting (for reasons given below) and there are no user-managed lists (see the last 7.2.3 paragraph).

To understand the interaction among Locations, Resources, and Variables, we need to consider the model definition framework of ProModel. The entity-flow part of ProModel is specified by the modeler via an ordered collection of *Process Steps* that make up a *Process Table*. Every Process Step includes 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 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 at this Location."

A Process Step is composed of *Operation Logic* and/or *Routing Logic* components. Competition among Entities for non-transportation Resources is spelled out in the Operation Logic. Competition among Entities for Locations and transportation Resources is spelled out in the Routing Logic. Routing Logic is acted upon after Operation Logic has been acted upon.

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 among 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 be represented in two or more delay lists of the same type simultaneously. As a result, ProModel does not require a polling mechanism for modeling certain Boolean conditions. (The Entity representations are deleted from all relevant delay lists when a delay-ending condition comes about.)

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 user-managed 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, to be executed by another Entity at the destination Location. This explicit triggering makes these special lists *resemble* user-managed lists. But because the lists and conditions are just for Locations and are not custom-managed, we consider the waiting to be related waiting and the lists to be delay lists, not user-managed lists.

7.3 GPSS/H

GPSS/H equivalents of the generic terms in the preceding sections are given in Table 3. For example, GPSS/H uses *Blocks* to specify operations for *Transactions*.

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
User-Managed List	User Chain

Table 3: GPSS/H Terminology

7.3.1 The Current Events Chain

As in SIMAN, the current events list is named the Current Events Chain in GPSS/H. A striking difference between GPSS/H and SIMAN is that by default in GPSS/H, Condition-Delayed Transactions (Xacts) are commingled with Ready-State Transactions on the CEC. For such Xacts, the CEC itself can be thought of as a single global GPSS/H delay list.

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

A characteristic of GPSS/H is that Transactions on the CEC are ranked FIFO within Priority Class. (Priority Class is a Transaction attribute.) This reflects the CEC's global-delay-list function.

Like other types of delay lists and unlike other types of current events lists, the GPSS/H CEC is frequently *not* empty whenever an EMP ends.

7.3.2 The Scan Phase

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. (See Schriber 1991.)

GPSS/H starts a Scan Phase with the Transaction (Xact) at the head of the CEC and tries to move that candidate-Xact 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 Xact is in a Condition-Delayed State and GPSS/H *leaves the candidate on the CEC* and examines the sequential CEC Xact. If entry is not denied, then the candidate becomes the active Xact (without being removed from the CEC) and begins executing Blocks.

If the active Transaction tries to execute a Block and entry is denied, the Transaction shifts to the Condition-Delayed State and remains on the CEC. GPSS/H then resumes scanning the CEC for the next active Transaction. However, because of possible state changes precipitated by the previously active Transaction's Block execution(s), the scan will either continue sequentially or *restart* (see 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 the Ready State at the instant of examination implies that all of these Transactions are fundamentally in a polled wait condition.

7.3.3 Restarting the Scan

GPSS/H has an internal status change flag (SCF) that is set to *true* whenever a *unique blocking condition* (see Section 7.3.4) is resolved. If the SCF is *true* when the active Transaction ceases to be active, 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 of the CEC continues with the sequential CEC Xact.

Scan restarts occur because there may be Transactions at or near the top of the CEC that should be given first crack at moving in response to the resolution of a unique blocking condition. The net effect of scan restarts and CEC Xact ranking within Priority Class is to provide FIFO-within-Priority-Class queuing in GPSS/H for condition-delayed Xacts resident on the CEC.

The active Xact can execute a YIELD (synonym: BUFFER) Block in GPSS/H to return itself temporarily to the Ready State and force an *immediate* scan restart. The restarted scan will eventually re-encounter the yielding Xact at the same simulated time, which will then again become active. The ability of an Xact to yield control deliberately but only temporarily to one or more other Xacts is quite useful in discrete-event modeling.

7.3.4 Related Waiting on the CEC

State changes involving unique blocking include the transition of a Facility (server) 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 an on-or-off 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 possible as well.)

Scan restarts imply extra processing demands while GPSS/H re-encounters and re-evaluates CEC Transactions. To offset this each Transaction has a Scan Skip Indicator (SSI) that flags those Transactions waiting for unique blocking conditions to be resolved. The SSI flag is checked before an attempt is made to move a candidate-for-active Transaction into its next Block, allowing the scan to quickly skip over such Condition-Delayed Transactions.

An Xact's SSI is cleared automatically at the moment of resolution of the unique blocking condition which has been forcing the Xact to wait. Internal delay lists are used to track which Xacts' SSIs need to be cleared in response to a given state change. These lists *are* related to an underlying condition, so the polled-waiting nature of the GPSS/H CEC scan is in fact a hybrid polled/related approach in the case of unique blocking.

7.3.5 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 their appropriate place on the CEC.

GPSS/H does not use internal entities to model downtimes. GPSS/H models downtimes (and some other control conditions as well) with actual Xacts. These are ordinary Xacts (external entities) that go through the ordinary Time-Delayed State to simulate time-between-failures and time-to-repair.

7.3.6 User Chains

GPSS/H implements the Dormant State with User Chains, which are user-managed lists of Xacts. After a Transaction puts itself onto a User Chain (by executing a LINK Block), it can only be removed by another Transaction (which triggers the removal by executing an UNLINK Block). If UNLINK execution transfers one or more Dormant-State Transactions to Ready State, the SCF will be made *true* (to trigger a scan restart) so these CEC newcomers will have their turn to become active before the next CUP. User Chains can achieve performance improvements over CEC-based queuing because User Chains (like delay lists in other tools) need never be scanned except when an UNLINK is executed.

8 WHY IT MATTERS

8.1 Overview

We now describe three situations that reveal some of the practical differences in implementation particulars among SIMAN, ProModel and GPSS/H. (Space restrictions limit us to three situations.) We then conclude with comments on how knowledge of software internals is needed to make effective use of software checkout tools.

8.2 Trying to Re-capture a Resource Immediately

Suppose a part releases a machine, then immediately re-competes for the machine (e.g., RELEASE followed by SEIZE in SIMAN or GPSS/H; FREE or USE followed by GET or USE in ProModel). The objective is to let a more highly qualified waiting part be the next to capture the machine; in the absence of such a part, the releasing part itself is to re-capture the machine.

Of interest here is the order of events following the giving up of a server. There are at least three alternatives: (1) Coupled with the giving up of the server is the immediate choosing of the next user of the server, without the releasing entity having yet become a contender for the server. (2) The choosing of the next user of the server is deferred until the releasing entity has become a contender. (3) "Neither of the above"; that is, without paying heed to other contenders, the releasing entity recaptures the server immediately.

SIMAN, ProModel, and GPSS/H respectively implement the first, second and third alternatives by default. And so one or another alternative is in effect in the tools considered here, reflecting differing implementation choices made by the software designers.

Note that these alternatives are not intrinsically either "right" or "wrong." The modeler must be aware of the alternative in effect and work with it to produce the desired outcome. (If a modeler is unaware of the alternative in effect in the simulation software being used, it is possible to model a given situation with an unintended effect and perhaps not even become aware of this fact.)

8.3 The First in Line is Still Delayed

Suppose two Condition-Delayed entities are waiting in a list because no units of a particular resource are idle. Suppose the first entity needs *two* units of the resource, whereas the second entity only needs *one* unit. Now assume that one unit of the resource becomes idle. The needs of the first list entity cannot yet be satisfied, but the needs of the second entity can. What will happen?

There are at least three possible alternatives: (1) Neither entity claims the idle resource unit. (2) The first entity claims the one idle resource unit and waits for a second unit. (3) The second entity claims the idle resource unit and goes immediately on its way.

As in Section 8.2, each of these alternatives comes into play in the tools considered here. SIMAN (SEIZE), ProModel (GET or USE), and GPSS/H (ENTER or TEST) respectively implement the first, second and third alternatives by default.

6.4 Yielding Control

Suppose the active entity wants to give control to one or more Ready-State entities, but then needs to become the active entity again before the simulated clock has been advanced. This might be useful, for example, if the

active entity has opened a switch permitting a set of other entities to move past a point in the model, and then needs to re-close the switch after the forward movement has been accomplished. (Perhaps a group of identically-flavored cartons of ice cream is to be transferred from an accumulation point to a conveyor leading to a one-flavor-per-box packing operation.)

In SIMAN, the effect can be accomplished approximately with a DELAY that puts the active Entity into a Time-Delayed State for an arbitrarily short but non-zero simulated time.

In ProModel, "WAIT 0" can be used to put the active Entity back on the FEL. It will be returned later (at the same simulated time) by the CUP to the Active State.

In GPSS/H, the active entity can execute a YIELD (BUFFER) Block to move from the Active State to the Ready State and restart the CEC scan. Higher-priority Xacts on the CEC can then become active, one by one, before the control-yielding Xact itself again becomes active at the same simulated time. (If all relevant Xacts have the same priority, a "PRIORITY PR,YIELD" Block can be used to reposition the active Xact behind equal-priority Xacts on the CEC, shift the active Xact to the Ready State, and restart the scan of the CEC.)

6.5 Interactive Model Verification

We now comment briefly on why a detailed understanding of "how simulation software works" 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 used to make production runs.

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 the current and future events lists and the delay and user-managed lists as well as other aspects of the model. These activities yield valuable insights into model behavior for the modeler who knows the underlying concepts. Without such knowledge, the modeler might not take full advantage of the interactive tools provided by the software or, worse yet, might even avoid using the tools.

ACKNOWLEDGMENTS

Much of the information in the original version of this paper was derived from conversations with software-vendor personnel. The authors gratefully acknowledge the support provided by David T. Sturrock, Deborah A. Sadowski, C. Dennis Pegden and Vivek Bapat, all of Systems Modeling Corporation; Charles Harrell and Eric Du, of ProModel Corporation; Kerim Tumay (now of CACI Products Company); and Robert C. Crain and James O. Henriksen, of Wolverine Software Corporation.

REFERENCES

Baird, S. P., and J. J. Leavy. 1994. Simulation Modeling Using ProModel for Windows. In *Proceedings of the 1994 Winter Simulation Conference*, 527-532. LaJolla, California: Society for Computer Simulation.

Banks, J., J. S. Carson, and J. N. Sy. 1995. *Getting Started with GPSS/H*, Second Edition. Annandale, Virginia: Wolverine Software Corporation.

Banks, J., B. Burnette, H. Kozloski, and J. Rose. 1995. *Introduction to SIMAN V and Cinema V*. New York, New York: John Wiley & Sons.

Crain, R. C. 1995. GPSS/H Release 3. In *CHECKPOINT*. Vol. 11, No. 1. Annandale, Virginia: Wolverine Software Corporation.

Crain, R. C., and D. S. Smith. 1995. Industrial Strength Simulation Using GPSS/H. In *Proceedings of the 1995 Winter Simulation Conference*, 487-493. LaJolla, California: Society for Computer Simulation.

Henriksen, J. O. 1995. An Introduction to SLX. In *Proceedings of the 1995 Winter Simulation Conference*, 502-509. LaJolla, California: Society for Computer Simulation.

Pegden, C. D., R. E. Shannon, and R. P. Sadowski. 1995. *Introduction to Simulation Using SIMAN*, Second Edition. New York, New York: McGraw-Hill.

Profozich, D. M., and D. T. Sturrock. 1995. Introduction to SIMAN/Cinema. In *Proceedings of the 1995 Winter Simulation Conference*, 515-518. LaJolla, California: Society for Computer Simulation.

ProModel Corporation. 1995. *ProModel Version 3 User's Guide*. Orem, Utah: ProModel Corporation.

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

Swain, J. J. 1995. Simulation Survey: Tools for Process Understanding and Improvement. *OR/MS Today*, August '95, 64-79, Baltimore, Maryland: INFORMS.

AUTHOR BIOGRAPHIES

DANIEL T. BRUNNER is President of Systemflow Simulations, Inc., a services firm active in manufacturing, material handling, distribution, transportation, health care, computer systems, and mining. He received a B.S.E.E. from Purdue University and an MBA from The University of Michigan. He has served as Winter Simulation Conference Publicity Chair (1988) and Business Chair (1992), and is General Chair for 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 a number of subjects while doing research and consulting in discrete-event simulation. He is a member of ASIM (the German-language simulation society), DSI, IIE, and INFORMS.