

## ON-LINE DEMONSTRATION OF SIMAN

C. Dennis Pegden  
Associate Professor, The Pennsylvania State  
University

Mike Joost  
Associate Professor, North Carolina State  
University

### Abstract

This paper provides an overview of the SIMAN simulation language. The SIMAN language is FORTRAN based and runs on 16 bit microcomputers as well as large mainframe computers. Following the presentation of the language concepts, the program will be demonstrated on a personally owned microcomputer. The written paper provided here is concerned only with the modeling concepts included in SIMAN.

### INTRODUCTION

SIMAN is a new combined discrete-continuous SIMulation ANalysis language for modeling general systems (1). In addition to containing powerful general purpose features which are not currently found in other languages, it also incorporates a number of special purpose features for modeling manufacturing systems. These special purpose features are particularly useful in modeling the material handling component of a complex manufacturing system. In the following sections an overview of SIMAN is given.

### THE SYSTEM--THEORETIC FRAMEWORK

The SIMAN modeling framework is based on the system theoretic concepts developed by Zeigler and Oren (2,3). Within this framework, a fundamental distinction is stressed between the system model and the experimental frame. The system model defines the static and dynamic characteristics of the system. In comparison, the experimental frame defines the experimental conditions under which the model is run to generate specific output data. For a given model, there can be many experimental frames resulting in many sets of output data. By separating the model structure and the experimental frame into two distinct elements, different simulation experiments can be run by changing only the experimental

frame. The system model remains the same.

Within this framework, component models based on three distinct modeling orientations can be combined in a single system model. For discrete change systems either a process or event orientation can be used to describe the model. Continuous change systems are modeled with algebraic difference/differential equations. The combination of these orientations can be used to represent combined discrete-continuous systems.

Given the system model and the experimental frame, the SIMAN simulation program generates an output file which records the model state transitions as they occur in simulated time. The data in the output file can then be used for various data analyses such as data truncation and compression, and the formatting and display of histograms, plots, tables, etc.

Within the SIMAN framework, the data analysis and display function are done after the development and running of the simulation program and are completely distinct from it. One output file can be subjected to many different data treatments without re-executing the simulation program. Data treatments can also be applied to sets of output files--this is useful when performing an analysis based on multiple runs of a model or when comparing the system response of two or more models.

### Modeling Orientations

The primary modeling orientation for discrete change systems is the process orientation, in which the model is constructed by depicting the functional operations of the systems as a block diagram. The block diagram is a linear top-down sequence of blocks which represents specific process functions such as time delays and queues. The block diagram may be specified in either an interactive mode in which the diagram is constructed graphically on a computer terminal, or a batch mode in which the diagram is described with batch input statements. In this paper we restrict our attention to the latter.

A second modeling orientation for discrete change systems, the event orientation, may be used to augment or replace the block diagram component.

Proceedings of the 1982  
Winter Simulation Conference  
Highland \* Chao \* Madrigal, Editors

82CH1844-0/82/0000-0523 \$00.75 © 1982 IEEE

The event component consists of a set of user-written FORTRAN subroutines which contain the mathematical-logical expressions that define the instantaneous state transitions occurring at each event time. The event subroutines typically use calls to subprograms contained in the SIMAN Subprogram Library. These subprograms perform standard event modeling functions such as file manipulating, event scheduling, observation recording, etc. The simulation is controlled by advancing time and making calls to the appropriate event subroutines at the correct simulated time.

The continuous component in a SIMAN system model results from coding the necessary algebraic difference/differential equations in FORTRAN within subroutine STATE. When differential equations are included, SIMAN automatically integrates the derivatives over time to obtain the system response.

### THE SOFTWARE STRUCTURE

As shown in Figure 1, a SIMAN simulation is divided into three distinct activities: system model development, experimental frame development, and data analysis. Within these three activities, the SIMAN software consists of five individual processors which interact through four data files.

1. The model processor is used to construct a block diagram model. The data file that is generated is called the model file.
2. The experiment processor is used to define the experimental frame for the system model. The data file that is generated is called the experiment file.

3. The link processor combines the model file and the experiment file to produce the program file.
4. The program file is input to the run processor which executes the simulation runs and writes the results on the output file. If the system model includes an event or continuous model, the user-written FORTRAN subroutines are linked to the run processor before the simulation runs are executed.
5. The output processor is used to analyze, format and display the data contained in the output file.

The five independent processors within the SIMAN software simplify the framework by separating the distinct functional activities of simulation. This breakdown is also practical, since only one of the five processors is executing at one time, and computer memory requirements are substantially reduced. In addition, you can independently create and save the model files, experiment files, program files, and output files on disk or tape for future use.

### BLOCK DIAGRAM MODELS

Block diagrams are the primary means for modeling discrete systems in SIMAN. These diagrams are linear top-down flowgraphs which show the movement of entities through the system. The shape of the individual blocks indicates their function. The sequencing is depicted by arrows which control the flow of entities from block to block through the entire diagram.

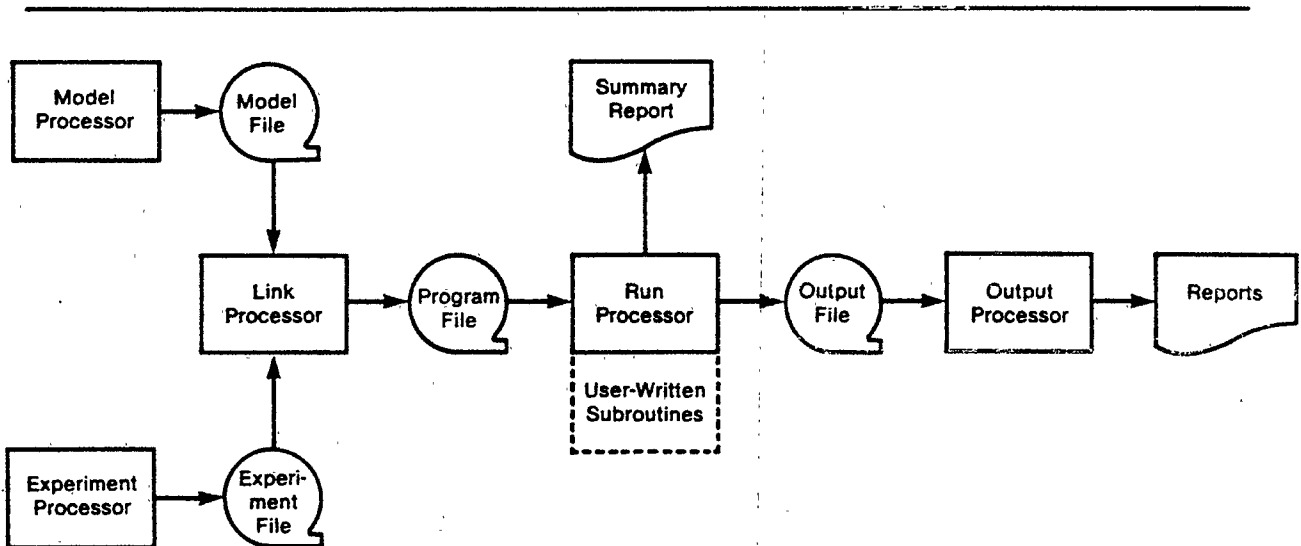





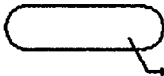

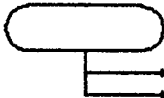
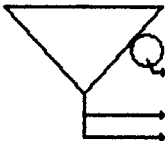
Figure 1. SIMAN Software Organization

These entities are used to represent "things" such as workpieces, information or people which flow through the real system. Each entity may be individualized by assigning attributes to describe or characterize it. For example, an entity representing a workpiece might have attributes corresponding to the due date and processing times. As the entities flow from block to block, they may be delayed, destroyed, combined with other entities, etc., as determined by the function of each block.

There are ten different basic block types in SIMAN. The symbol and function for each of these blocks are summarized in Table 1.

### The Block Function Name

Each block function in SIMAN is referenced by a block function name. In the use of the QUEUE, STATION, BRANCH, PICKQ, QPICK, SELECT and MATCH blocks, each block type performs only one function and the block function name is the same as the basic block name. However, in the case of the OPERATION, HOLD and TRANSFER blocks, each basic block type performs several different functions. The block function name for each of these blocks is the operation type, hold type or transfer type specified as the first operand of the block. We will frequently refer to a block by its function

Name	Symbol	Function
OPERATION		The OPERATION block is used to model a wide range of processes such as time delays, attribute assignments, etc. Also see Table 1-2.
TRANSFER		The TRANSFER block is used to model transfers between stations via material handling systems. Also see Table 1-4.
HOLD		The HOLD block is used to model situations in which the movement of an entity is delayed based on system status. The HOLD block must be preceded by a queueing facility to provide a waiting space for delayed entities. Also see Table 1-3.
QUEUE		The QUEUE block provides a waiting space for entities which are delayed at following HOLD or MATCH blocks.
STATION		The STATION block defines the interface points between model segments and the material handling systems.
BRANCH		The BRANCH block models the conditional, probabilistic and deterministic branching of entities.
PICKQ		The PICKQ block is used to select from a set of following QUEUE blocks.

(Continued on next page)

Table 1. Basic Block Types

(Continued from previous page)

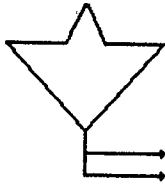
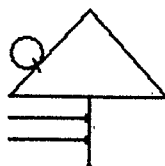
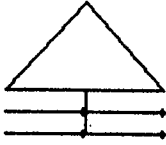
Name	Symbol	Function
SELECT		The SELECT block is used to select between resources associated with a set of following OPERATION blocks.
QPICK		The QPICK block is used to select from a set of preceding QUEUE blocks.
MATCH		The MATCH block delays entities in a set of preceding QUEUE blocks until entities with the same value of a specified attribute resides in each QUEUE.

Table 1. Basic Block Types

name--for example, we will refer to the OPERATION block with the DELAY operation type as the DELAY block.

#### Types of Block Functions

The OPERATION, HOLD and TRANSFER blocks are further subdivided into several different block functions depending upon their operation type, hold type or transfer type. These types (specified as the first operand of the block) consist of a verb describing the function of the block. For example, the operation type CREATE specifies that the block is to create entities; the operation type ASSIGN specifies that the block is to assign a value to an attribute or variable; and the operation type DELAY specifies that the block is to delay entities. A summary of the various operation types, hold types and transfer types included in SIMAN is given in Tables 2, 3 and 4.

All of the basic block types, including the OPERATION, HOLD, and TRANSFER types, have operands which control the function of the block. For example, the CREATE block has operands which prescribe the time between batch arrivals, the number of entities per batch, and the maximum number of batches to create. The operands for the blocks are described in detail in the next sections.

#### Inputting the SIMAN Model File

As described earlier, block diagrams may be entered into the SIMAN model file in either an interactive graphics mode or a batch mode using input statements. In the interactive graphics mode, the block diagram model is constructed directly on the computer terminal with assistance from real time error checking. In comparison, the batch mode (using input statements) provides an equivalent but more concise form for describing a block diagram model in a batch computing environment. Regardless of the method used to enter it, the block diagram is stored within the SIMAN model file using a special data structure which allows either the statement form or the graphic form of the model to be recreated. Hence, a model entered using input statements can be later displayed on a graphics terminal in graphics form and vice versa.

When block diagrams are entered graphically at a computer terminal, each added block is automatically inserted at the bottom of the diagram, causing the preceding blocks to eventually scroll up and off the top of the screen. These blocks are automatically aligned vertically. Blocks may be assigned labels consisting of up to eight alphanumeric characters (starting with a letter) which are appended to the lower left-hand side of the

	Name	Description
General Function	ASSIGN	Assign values to attributes and variables.
	CREATE	Create batch arrivals to the system.
	DELAY	Delay an entity by a specified time.
	DETECT	Detect state events associated with continuous variables.
	EVENT	Cause a specified event to occur.
	FINDJ	Find the value of the index J meeting a specified condition.
	SIGNAL	Send a signal to end the delay for an entity.
	SPLIT	Split a group into its individual members.
Resource Function	ALTER	Change the capacity of a specified resource.
	RELEASE	Release units of a specified resource.
Material-Handling Function	ACTIVATE	Set the status of a specified transporter to active.
	EXIT	Exit a specified conveyor device.
	FREE	Exit a specified transporter device.
	HALT	Set the status of a specified transporter to inactive.
	START	Set the status of a specified conveyor to active.
	STOP	Set the status of a specified conveyor to inactive.
File Function	COPY	Copy the attributes of an entity at a specified QUEUE.
	REMOVE	Remove an entity from a specified QUEUE.
	SEARCH	Search a QUEUE for an entity meeting a specified condition.
Statistics Function	COUNT	Increment a specified counter.
	TALLY	Record an observation of a specified value.

Table 2. Operation Types

block. These labels may be used in branching and for referencing by other blocks. Blocks are automatically assigned sequence numbers by SIMAN and these appear adjacent to the blocks on the far left side of the screen. These sequence numbers are used for block editing, while the right half of the screen is reserved for user comments.

### Block Modifiers

The blocks are connected using the connector symbols + and L, LABEL. The first connector type, the +, is used for directing sequential flow of entities from one block to the next. The second connector type, the L, LABEL, directs non-sequential flows where LABEL specifies the label of the block to where the entity is to be sent next. If neither connector is appended to a block, the special | is added to indicate that entities departing the block are destroyed. Connectors may

not be appended to TRANSFER blocks since the flow of entities from these blocks is controlled by the block function.

In addition to the connector symbols, the special mark symbol <MA may be appended to the right-side of any block to denote the marking of attribute number MA with the arrival time of the entity to the block. The use of the marking option will be explained in detail later. The mark symbol and the connector symbols discussed above are referred to as block modifiers.

The general block description for batch input statements is divided into five sections consisting of the block label (if any), the block name, the block operands, the block modifiers (if any), and the block comment field (if any). The block label is entered anywhere within the first eight columns of the input record with the remaining description starting in column ten or after. The operands are

	Name	Description
Condition Function	SCAN	Hold the entity until a specified condition is met.
	WAIT	Hold the entity until a specified signal is received from a SIGNAL block.
Resource Function	SEIZE	Hold the entity until the required number of units of a resource are idle and are allocated to the entity.
	PREEMPT	Hold the entity until one resource unit is allocated to the entity. The entity may preempt a resource currently being used.
Material-Handling Function	ACCESS	Hold the entity until a specified number of consecutive conveyor cells are available and allocated to the entity at the accessing station location.
	REQUEST	Hold the entity until a transporter device is allocated to the entity and arrives to the requesting station location.
Set Function	COMBINE	Hold the entity until a specified number of entities reside in the preceding QUEUE block. When this occurs, the waiting entities are combined into a permanent set and a representative of the set is created. The original entities in the set are destroyed.
	GROUP	Hold the entity until a specified number of entities reside in the preceding QUEUE block. When this occurs, the entities are grouped into a temporary set and a representative of the set is created. The original entities in the set are retained and can be recovered using the SPLIT block.

Table 3. Hold Types

	Name	Description
Material-Handling Function	CONVEY	Convey the entity to a specified station via a conveyor. The transmit time is determined by the distance between the station along the conveyor and the speed of the conveyor.
	ROUTE	Route the entity to a specified station. The transmit time is specified as an operand of the block.
	TRANSPORT	Transport the entity to a specified station via a transporter. The transmit time is proportional to the distance between stations.

Table 4. Transfer Types

divided into line segments which are ended with the special character ":" where each line segment corresponds to a given line of operands within a block. Any operand may be defaulted by simply omitting the entry from the line segment. The last operand line is followed by the block modifiers which may be one or more of the keywords DISPOSE, MARK (MA), or NEXT (LABEL) separated by commas. The DISPOSE modifier causes the departing entities to be destroyed, the MARK (MA) modifier prescribes the marking of attribute MA with the arrival time to the block, and NEXT (LABEL) specifies a non-sequential flow to the block specified by LABEL. If the DISPOSE and NEXT (LABEL) modifiers are omitted, sequential flow to the next block is assumed. The end of all operands and modifiers is indicated by the special character ";" which is

followed by the comment field.

#### A Sample SIMAN Model

The following example of a TV inspection and adjustment process (4) illustrates the general arrangement of blocks on the computer graphics terminal and the format for the corresponding input statements (see Figures 2 and 3). In this model, the vertical control setting of TV sets is tested at an inspection station. If a set is found to be functioning improperly, it is routed to an adjustment station. After adjustment, the TV set is sent back to the inspection station where setting is again inspected. TV sets passing inspection, whether for the first time or after one or

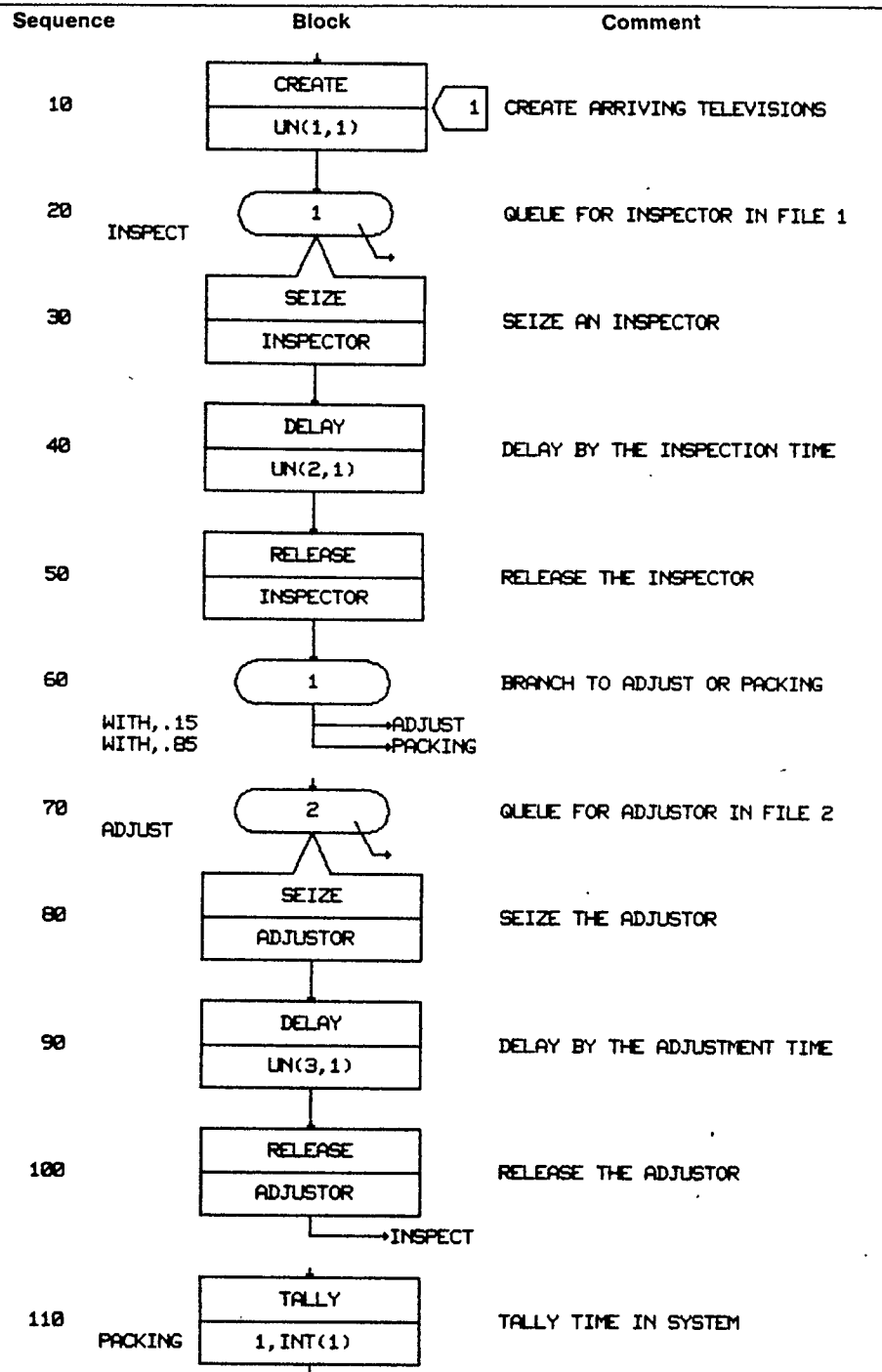


Figure 2. Block Diagram Model for Example 1A

more routings through the adjustment station, are sent to a packing area.

Since the functional operations of the blocks used in this model have not yet been fully described, a detailed explanation of the block diagram model shown in Figure 2 will not be attempted at this point. This example is included with the intent of providing the reader with an overview of the format for a SIMAN block diagram.

The corresponding input statements for this

same model for use within a batch mode are shown in Figure 3. Again the intent here is not to provide a detailed discussion of the input statements but to illustrate the general format conventions.

The model in Figure 3, whether entered in the graphic or statement mode, would be combined with an experimental frame which would prescribe the resource capacities, distribution parameters, random number seeds, run length, etc. Figure 4 depicts the input statements for one possible experimental frame for use with this model.

```

BEGIN;
10      CREATE:UN(1,1):MARK(1);      CREATE ARRIVING TELEVISIONS
20 INSPECT  QUEUE,1;                QUEUE FOR INSPECTOR IN FILE 1
30      SEIZE:INSPECTOR;            SEIZE AN INSPECTOR
40      DELAY:UN(2,1);              DELAY BY THE INSPECTION TIME
50      RELEASE:INSPECTOR;          RELEASE THE INSPECTOR
60      BRANCH,1:
      WITH, .15,ADJUST:
      WITH, .85,PACKING;
70 ADJUST  QUEUE,2;                BRANCH TO ADJUST OR PACKING
80      SEIZE:ADJUSTOR;            QUEUE FOR ADJUSTOR IN FILE 2
90      DELAY:UN(3,1);            SEIZE THE ADJUSTOR
100     RELEASE:ADJUSTOR:NEXT(INSPECT); DELAY BY THE ADJUSTMENT TIME
110 PACKING TALLY:1,INT(1):DISPOSE; RELEASE THE ADJUSTOR
      TALLY TIME IN SYSTEM
END;

```

Figure 3. Input Statements for Example 1A

```

BEGIN;
10 PROJECT,TV INSP AND ADJT,PEGDEN,4/25/81;
20 DISCRETE,30,1,2;
30 PARAMETERS:1,3.5,7.5:2,6.,12:3,20,40;
40 RESOURCES:1,INSPECTOR,2:2,ADJUSTOR;
50 TALLIES:1,TIME IN SYSTEM;
60 DSTAT:1,NQ(1),INSP QUEUE:2,NQ(2),ADJT QUEUE:
      3,NR(1),INSP UTIL:4,NR(2),ADJT UTIL;
70 TRACE;
80 REPLICATE,1,0,100;
END;

```

Figure 4. Experimental Frame for Example 1A

## DISCRETE EVENT MODELS

An event orientation is included in SIMAN to augment the block diagram model of a discrete change system. The event orientation can be used to develop specialized logic which is not provided by the set of standard block functions. In this section, the procedures for including discrete events within a model are described.

Three distinct types of events can be incorporated: the block event, the time event and the state event. The block event is initiated by an entity arrival to an EVENT block included in the block diagram model. The time event is scheduled to occur at a specified point in simulated time. Time events are set by a special event scheduling subroutine (named SCHED) included in SIMAN. The state event is initiated by a continuous submodel when a specified continuous variable crosses a prescribed threshold. The crossing conditions for a state event are specified by the modeler in the experimental frame. Although the mechanism for initiating the three event types differ, the procedures are identical for describing the mathematical-logical relationships which define the instantaneous change in system status resulting from the occurrence of the event. Hence the rest of this section applies to all three event types.

The mathematical-logical relationships defining each event are incorporated in user-coded FORTRAN subroutines. Each event is coded as a separate subroutine and typically uses calls to one or more subprograms provided in the SIMAN

Subprogram Library. The subprograms perform standard event functions such as event scheduling, file manipulations and statistics recording. These subprograms are summarized in Table 5.

Within the SIMAN event orientation, all references to entities are made using the entity record location, denoted by L. The record location of an entity is a FORTRAN integer variable which can be assigned an arbitrary name by the user. This variable is a pointer to the starting address of the entity record within an internal SIMAN storage array. The entity record is a data record for maintaining the current values for the entity attributes and related information. During a simulation run, entity records are dynamically allocated from, and returned to, the pool of available records as entities enter and leave the system. An entity entering the simulated system is created by assigning it a record location from the pool of available records. This is done by calling subroutine CREATE(L) where the argument L denotes the arbitrary variable name assigned to the record location. The entity record location remains fixed and assigned to that entity as long as the entity remains in the simulated system. The entity record of a departing entity is returned to the pool of available records by calling subroutine DISPOS(L).

The interface between SIMAN and the user-coded event subroutines is provided by user-coded subroutines EVENT(L,I). The argument I is the event code which is a positive integer assigned to each event included in a SIMAN model. This event



Subroutine	Description
CREATE(L)	Creates an entity by assigning an entity record location to the integer variable L.
DISPOS(L)	Disposes an entity by returning the entity record at location L to the pool of available records.
SCHED(L,I,DT)	Schedules event number I to occur in DT time units from the current time for the entity with record location L.
INSERT(L,IFL)	Inserts the entity with record location L into file IFL. The relative rank of the entity in the file is determined by the ranking rule specified for the file in the experimental frame.
REMOVE(L,IFL)	Removes the entity with record location L from file IFL.
COPY(L,A)	Copies the attributes of the entity with record location L to the real array A.
ASSIGN(L,A)	Assigns the values in the real array A to the attributes of the entity with record location L.
SETA(L,N,VAL)	Sets the value of the Nth attribute of the entity with record location L to the real value specified as the argument VAL.
SETM(L,NS)	Sets the station number for the entity with record location L to the integer value specified as the argument NS.
SETP(NS,NP,VAL)	Sets parameter number NP of parameter set NS to the real value specified as the argument VAL.
TALLY(N,VAL)	Records the real value specified by the argument VAL as an observation of tally variable number N.
COUNT(N,INC)	Increments counter number N by the integer value specified as the argument INC.
ENTER(L,NS)	Enters the entity with record location L into STATION block number NS.
QUEUE(L,IFL)	Schedules the entity with record location L to arrive at the QUEUE block using file number IFL.
Function	Description
A(L,N)	The value of the Nth attribute for the entity with record location L.
M(L)	The station number of the entity with record location L.
NQ(IFL)	The current number of entities in file IFL.
LFR(IFL)	The record location of the first entry in file IFL.
LLR(IFL)	The record location of the last entry in file IFL.
LRANK(NR,IFL)	The record location of the entry with rank NR in file IFL.
LPRED(L)	The record location of the predecessor to the entry at location L.
LSUCC(L)	The record location of the successor to the entry at location L.

Table 5. SIMAN Subprogram Library (Excluding Random Sample Functions)

is used for all references to the event. The execution of an event is processed by a call to subroutine EVENT (L,I) where L is the associated entity record location and I is set by SIMAN to the appropriate event number to be executed. Subroutine EVENT (L,I) maps each event code onto a call to the user-written FORTRAN subroutine containing the logic for that event. The following is an example of subroutine EVENT (L,I) where Event 1 is coded in subroutine ARRIVE (JOB) and Event 2 is coded in subroutine DEPART (JOB). In the following example, the entity record location

is the FORTRAN variable JOB.

```

SUBROUTINE EVENT(JOB,I)
  GOTO (1,2),I
  1  CALL ARRIVE(JOB)
     RETURN
  2  CALL DEPART(JOB)
     RETURN
  END

```

There are two special events which are called by SIMAN during each run of a model. The first is

the start of simulation event which is user-coded in subroutine PRIME. The second is the end of simulation event which is user-coded in subroutine WRAPUP. These events are not assigned event codes since they are called directly by SIMAN at the start and end of each run, respectively. Dummy versions of both PRIME and WRAPUP are included in the SIMAN Subprogram Library so that user-written versions of either or both may be omitted.

The relationship between the SIMAN executive, the SIMAN Subprogram Library, and the user-written subroutines is illustrated in Figure 5.

### Discrete Event Model Example

To illustrate the SIMAN framework for event modeling, consider a model to simulate the processing of jobs on a single machine. Within our model, the state of the system will be the number of jobs in the system. The state will change when a new job arrives to the system, or a completed job departs from the system. Two events are required to model the changes in the system state: a job arrival event and a job departure event.

The development of a discrete event model for this problem requires the FORTRAN coding of an event subroutine to model the state transitions for each of the two events. This requires a thorough understanding of the routines within the SIMAN Subprogram Library, including the procedures for sampling from distributions. We will provide here only a brief discussion of the event routines

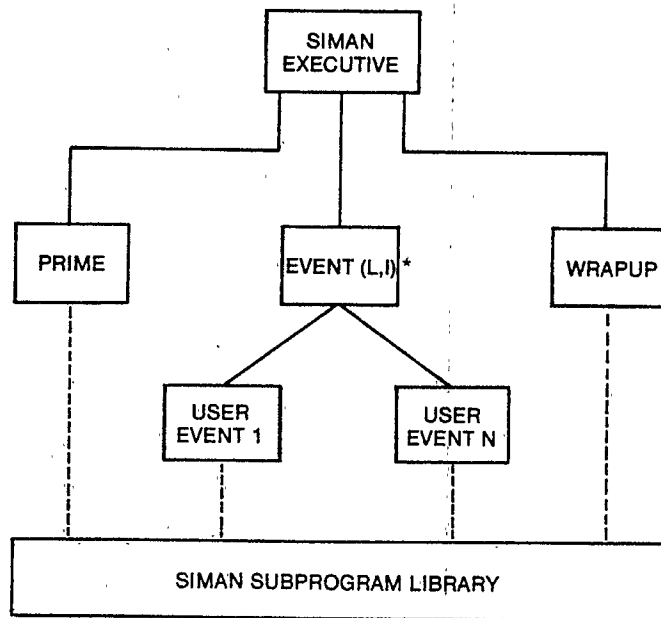
with the objective of providing the reader with an overview of the concepts and methods involved in discrete event modeling using SIMAN.

The FORTRAN coding for the two user-written routines and subroutine PRIME is shown in Figure 6. The job arrival event is coded as event number 1 in subroutine ARRIVE, and the job departure event is coded as event number 2 in subroutine DEPART. The linkage of the events is provided by subroutine EVENT (L,I) which is coded as shown in the previous example.

The model uses the FORTRAN variable STATUS to keep track of the busy/idle state of the machine, where a value of 1 denotes busy and a value of 0 denotes idle. Jobs in the system are modeled as entities with one attribute which is used to record the arrival time of the entity to the system. This attribute is assigned in subroutine ARRIVE and then used in subroutine DEPART to record statistics on the time in the system for the job.

Subroutine PRIME is used to establish the initial conditions for the simulation and to create and schedule the first arrival to the system. In this example the system is assumed to begin empty and idle, with the first job arrival scheduled for time 0.

Subroutine ARRIVE contains the FORTRAN coding for the job arrival event. The first action within the event is to reschedule the job initiating the event as the next job arrival and to create a record for the current job. In this way, only one job



L: Associated entity record location.  
I: Event type.

Figure 5. Subprograms Within SIMAN

```

SUBROUTINE PRIME
COMMON/SIM/D(50),DL(50),S(50),SL(50),X(50),DTNOW,TNOW,TFIN,J,NRUN
COMMON/MACHINE/STATUS
C
STATUS=0.0
CALL CREATE(NEWJOB)
CALL SCHED(NEWJOB,1,0.0)
RETURN
END

SUBROUTINE ARRIVE(NEWJOB)
COMMON/SIM/D(50),DL(50),S(50),SL(50),X(50),DTNOW,TNOW,TFIN,J,NRUN
COMMON/MACHINE/STATUS
C
CALL SCHED(NEWJOB,1,EX(1,1))
CALL CREATE(JOB)
CALL SETA(JOB,1,TNOW)
IF (STATUS.EQ.1) THEN
    CALL INSERT(JOB,1)
ELSE
    STATUS=1.0
    CALL SCHED(JOB,2,UN(2,1))
END IF
RETURN
END

SUBROUTINE DEPART(JOB)
COMMON/SIM/D(50),DL(50),S(50),SL(50),X(50),DTNOW,TNOW,TFIN,J,NRUN
COMMON/MACHINE/STATUS
C
TSYS=TNOW-A(JOB,1)
CALL TALLY(1,TSYS)
CALL DISPOS(JOB)
IF (NQ(1).GT.0) THEN
    JOB=LFR(1)
    CALL REMOVE(JOB,1)
    CALL SCHED(JOB,2,UN(2,1))
ELSE
    STATUS=0.0
END IF
RETURN
END

```

Figure 6. Subroutine PRIME and Two User-Written Routines

arrival event is scheduled to occur at any one time. However, a complete sequence of arrivals is generated. Next, attribute 1 of the arriving job is set to TNOW, which is a SIMAN variable denoting the current simulated time. A test is then made on the status of the machine. If the machine is busy, the job is placed in file number 1 to wait for the machine and a return is made to the SIMAN executive. Otherwise, the status is set to busy and the job departure event is scheduled, followed by a return to the SIMAN executive.

The job departure event is coded in subroutine DEPART. The first action is to record statistics on the time in the system for the departing job. The departing entity is then destroyed by returning its record to the available record pool by a call to subroutine DISPOS. Next, a test is made to determine if there are jobs waiting in file 1 for processing on the machine. If there are, the first waiting job is removed from file 1 and the departure event is scheduled for that job, followed by a return to the SIMAN executive. Otherwise, the machine status is set to idle and a return is made to the SIMAN executive.

Subroutines PRIME, ARRIVE and DEPART comprise the system model for this example. Prior to execution, the subroutines would be linked to the SIMAN run processor and combined with an experimental frame to prescribe the length and number of runs, the distribution parameters for the arrival and service process, etc.

#### CONTINUOUS COMPONENT MODELS

In a continuous simulation model, the state of the system is represented by dependent variables which change continuously over time. Examples of continuous change variables include the temperature of an ingot in a soaking pit furnace, or the concentration of a reactant within a chemical process. A continuous simulation model is constructed by defining equations for a set of state variables whose dynamic response simulates the real system.

The equations that define a continuous change system can be written as either algebraic, difference or differential equations. In algebraic equations, the state of the system is written

directly as a function of the other variables in the system. In a difference equation, the state is defined as a function of the other variables in the system and the state at a previous instant in time. Both algebraic and difference equations are referred to as state equations since the value of the state variables are computed directly by the equations. In contrast, differential equations define the state of the system by specifying the derivatives of the state variables. In the last case, SIMAN integrates the derivative over time to obtain the state of the system.

SIMAN may be used to model pure-continuous systems or combined discrete-continuous systems. In combined models, the continuous submodel may interact with the discrete submodel through either state events defined in the experimental frame and user-coded as event subroutines, or through the DETECT block included in the block diagram model. The values of the state variables and their derivatives are also available for logical testing or may be modified within the block diagram model or within event subroutines.

A continuous system is modeled in SIMAN by coding the algebraic, difference, and differential equations in FORTRAN within the user-coded subroutine STATE. Subroutine STATE is automatically called by SIMAN at small time intervals called steps to compute the response of the system over time. The value of the Ith state variable is maintained as variable S(I), and the derivative with respect to time of the Ith state variable is maintained as variable D(I). The values of these variables at the end of the last step are assigned to the variables SL(I) and DL(I), respectively. The current step size is denoted by the variable DTNOW and is automatically controlled by SIMAN based on accuracy parameters specified within the experimental frame. When differential equations are included within subroutine STATE, they are automatically integrated by SIMAN using the Runge-Kutta-Fehlberg integration algorithm to obtain the values of the state variables within an accuracy specified by the modeler in the experimental frame.

#### Continuous Model Example

To illustrate the SIMAN framework for continuous modeling, consider the population dynamics associated with the growth and decay of an infectious disease within a single population whose recovery results in immunity (5). The population consists of three groups: (1) those that are well but susceptible; (2) those that are sick; and (3) those that are cured and therefore immune. Although the system state actually changes discretely, we will assume that we can approximately the system with a continuous model given by the following differential equations:

$$\frac{d}{dt} (\text{WELL}) = -RI + \text{WELL} * \text{SICK}$$

$$\frac{d}{dt} (\text{SICK}) = RI * \text{WELL} * \text{SICK} - RR * \text{SICK}$$

$$\frac{d}{dt} (\text{CURED}) = RR * \text{SICK}$$

In this example, RI is the rate of infection and RR is the rate of recovery. We will assume that RI is .001 and RR is .07.

This model is implemented in SIMAN by coding the model equations in subroutine STATE. To do this, we must first make an equivalence between the problem variables and the SIMAN array S(.). In this example, we will let S(1) denote the number of well members, S(2) the number of sick members, and S(3) the number of cured members. The rate of change of the number in each group is therefore given by D(1), D(2), and D(3), respectively. The coding of the differential equations in subroutine STATE is shown below.

```

SUBROUTINE STATE
COMMON/SIM/D(50),DL(50),S(50),SL(50),...
DATA RI,RR/.001,.07/
D(1)=-RI+S(1)*S(2)
D(2)=RI*S(1)*S(2)-RR*S(2)
D(3)=RR*S(2)
RETURN
END

```

To execute the model, subroutine STATE would be compiled and linked to the run processor and combined with an experimental frame to define the initial conditions, output variables, run length, etc.

#### APPLICATIONS

SIMAN is a new simulation language and has not yet been distributed for general use. However, some application experience has been gained with the language at Tektronix, Inc. which has served as a preliminary test site for SIMAN. These applications include a power-and-free overhead monorail system, a miniloader storage and retrieval system, and a high volume conveyor line for CRT manufacturing.

#### REFERENCES

- (1) Pegden, C. D., Simulation Analysis Using SIMAN, manuscript in preparation, 1982.
- (2) Ziegler, B. P., Theory of Modelling and Simulation, John Wiley, New York, 1976.
- (3) Oren, T. I. and Ziegler, B. P., "Concepts for Advanced Simulation Methodologies," Simulation, 32, 3, pp. 69-82, 1979.
- (4) Schriber, T., Simulation Using GPSS, John Wiley, New York, 1974.
- (5) Korn, G. A. and J. V. Wait, Digital Continuous-System Simulation, Prentice-Hall, 1978.