1981 Winter Simulation Conference Proceedings
T.I. Ören, C.M. Delfosse, C.M. Shub (Eds.)

SLAM TUTORIAL

Claude Dennis Pegden
Associate Professor, Pennsylvania State University
Consultant, Pritsker and Associates, Inc.
University Park, PA 16802

A. Alan B. Pritsker
President, Pritsker and Associates, Inc.
Professor, Purdue University
West Lafayette, IN 47907

This paper provides an overview of the important features of the SLAM simulation language. The focus of the paper is the unified system-modeling framework of SLAM which allows systems to be viewed from process, event, or state variable perspectives.

1. INTRODUCTION

SLAM is a simulation language that allows for alternative modeling approaches. It allows systems to be viewed from a process, event, or state variable perspective. These alternate modeling world views are combined in SLAM to provide a unified systems modeling framework (1,4).

In SLAM, a discrete change system can be modeled within an event orientation, process orientation, or both. Continuous change systems can be modeled using either differential or difference equations. Combined discrete-continuous change systems can be modeled by combining the event and/or process orientation with the continuous orientation. In addition, SLAM incorporates a number of features which correspond to the activity scanning orientation.

The ability to construct combined network-event-continuous models with interactions between each orientation greatly enhances the modeling power of the systems analyst. In the following sections of this tutorial, a discussion of network, event, and continuous SLAM capabilities is given.

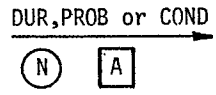
2. NETWORK CAPABILITIES

The process orientation of SLAM employs a network structure comprised of specialized symbols called nodes and branches in a manner similar to Q-GERT (2). These symbols model elements in a process such as queues, servers, and decision points. The modeling task consists of combining these symbols into a network model which pictorially represents the system of interest. In short, a network is a pictorial representation of a process. The entities in the system (such as people and items) flow through the network model. The pictorial representation of the system is transcribed by the modeler into an equivalent statement model for input to the SLAM processor.

Time delays associated with entities as they move through a network are prescribed for branches. Thus an activity that an entity engages in as it moves through a system is represented by a branch. In addition to the time delay referred to as the activity duration, other characteristics associated with a branch are: a probability or condition for an entity to take the branch; the number of parallel servers if the branch represents a service activity; and an activity number. Utilization statistics can automatically be obtained for any branch in a SLAM network. The graphical representation and

81CH1709-5/81/0000-0091$00.75

syntax for a branch is shown below.



Branches are used to separate nodes and each node in SLAM performs a distinct function. The fifteen node types included in SLAM are listed in Table 1.

Table 1. Types of Network Nodes in SLAM

Node Name	Function
ACCUM	Accumulates a set of entities into a single entity.
ALTER	Changes the capacity of a resource.
ASSIGN	Assigns values to entity attributes or global system variables.
AWAIT	Delays entities in a file based on the availability of a resource or the status of a gate.
CLOSE	Changes the status of a specified gate to be closed.
COLCT	Collects statistics and histograms on SLAM variables.
CREATE	Creates entities based on the specified arrival pattern.
FREE	Releases resource units seized at AWAIT and PREEMPT nodes.
GOON	A continuation (go on) or "do nothing" node.
MATCH	Delays entities in QUEUE nodes until a match condition occurs in which entities with the same value of a specified attribute are resident in every QUEUE node preceding the MATCH node.
OPEN	Changes the status of a specified gate to open.
PREEMPT	Preempts a resource seized at an AWAIT node or at a PREEMPT node with lower priority.
QUEUE	Delays entities in a file until a server becomes available. The QUEUE node is also used in conjunction with MATCH nodes.
SELECT	Selects among multiple queues and available servers based on the queue selection rule and server selection rule.
TERM	Destroys entities and terminates the simulation.

Five node types control the general flow of entities. This includes changing an entity's attribute values and the collection of statistics. These node types are: CREATE, ACCUMULATE (GOON), TERM, COLCT, and ASSIGN.

A general method for routing entities from these nodes involves specifying the maximum number of branches that can be taken from the node. By including a probability or condition on the branches emanating from the nodes, entity routing is specified. This same procedure is used for routing entities from other node types.

To simplify the process of representing a machine or machine group and its associated queue, SLAM includes a QUEUE node. Thus, a QUEUE node with a following branch is used to represent a processing station. When the processing station involves different processors, a SELECT node is used to select according to a prescribed rule which processor should be used when a choice exists. The SELECT node also is used to route entities to queues or to take them from queues. A MATCH node stops the flow of entities until an attribute match can be made with other entities flowing through the network. When a match occurs, the matched entities flow through the MATCH node.

Four nodes are included in SLAM that relate to the use of resources. A resource is a commodity that is required by an entity before it can continue on its route through the network. A resource in SLAM is allocated at an AWAIT node to an entity and it is not available for reallocation until the entity passes through a FREE node. When freed, the resource can be reallocated to an entity. Entities for which resources are not available wait for the resource at AWAIT nodes. Thus, the AWAIT node plays a similar function to a QUEUE node. These two node types are the only locations within a SLAM network where an implicit delay for an entity can occur. Resources can be preempted from an entity by other entities arriving to PREEMPT nodes. The syntax for the PREEMPT node specifies the disposition of the entity from which resources are preempted and the priority that enables one entity to preempt resources from another entity. The fourth node type associated with resources is the ALTER node which causes the available number of units of a resource type to be changed.

The last two nodes in SLAM are associated with gates. A gate is a mechanism for halting the flow of entities. If the gate is closed, entities are stored in AWAIT nodes until the gate is opened. The gate is opened by having an entity arrive to an OPEN node. When this occurs all entities being held up by the closed gate in AWAIT nodes are moved forward. An entity arriving to a CLOSE node, closes the gate.

As can be seen by the above discussion, the set of network elements in SLAM is concise yet powerful. In addition, functions exist for assigning samples of random variables for activity durations and to attribute values of entities. Also, SLAM variables for each of the network constructs is available to provide conditions for routing entities through the network. In this way, the path of an entity through the network can be made to be dependent on the status of a server, the number in a queue, resource availability, and the like. In the next section, a simple example of a SLAM network model is given.

3. A NETWORK MODEL IN SLAM

Consider a situation involving two types of jobs that require processing by the same server. The job types are assumed to form a single queue before the server. The network model of this situation is shown in Figure 1. The input statements corresponding to the network shown in Figure 1 are listed below.

```
NETWORK;
       CREATE,8,,,100;
       ASSIGN,ATRIB(1)=EXPON(7.1);
         ACTIVITY,,,QOFS;
       CREATE,12,,,50;
       ASSIGN,ATRIB(1)=EXPON(10.);
QOFS QUEUE(1);
         ACTIVITY/1,ATRIB(1)+RNORM(0.0,1.0);
       TERM;
       ENDNETWORK;
```

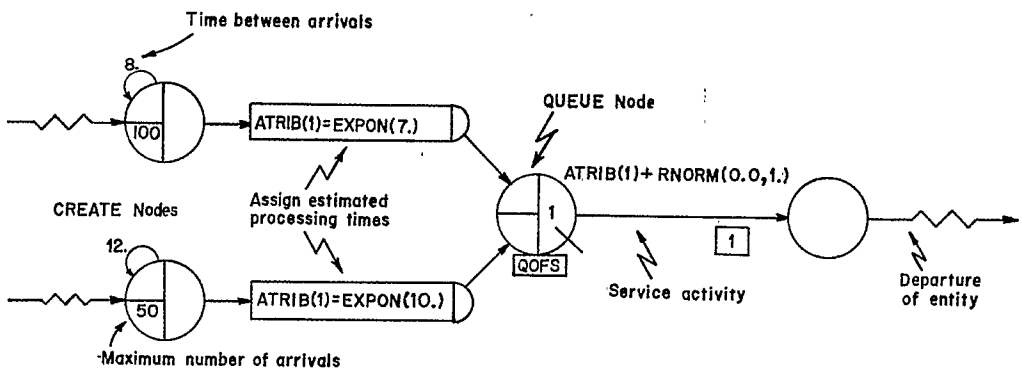


Figure 1. Network Model of a Multiple Entity, Single Server Queueing Situation

In this model, one type of entity is scheduled to arrive every 8 time units and only 100 of them are to be created. These entities have a service time estimated to be a sample from an exponential distribution with a mean time of 7. This service time is assigned to attribute 1 at an ASSIGN node. For the other type of entity, the time between arrivals is 12 time units and a maximum of 50 of these entities can be created. The estimated service time for each of these entities is exponentially distributed with a mean time of 10. Both types of entities are routed to a QUEUE node whose label is QOFS.

The server of the system is modeled as activity 1 where the service time is specified as attribute 1 plus a sample from a normal distribution. Thus, the actual processing time is equal to the estimated processing time plus an error term that is assumed to be normally distributed. This model might be used to represent a job shop in which jobs are performed in the order specified by the ranking rule specified for the QUEUE node.

4. DISCRETE EVENT CAPABILITIES

In the event orientation of SLAM, the modeler defines the events and the potential changes to the system when an event occurs. The mathematical-logical relationships prescribing the changes associated with each event type are coded by the modeler as FORTRAN subroutines. A set of standard subprograms is provided by SLAM for use by the modeler to perform common discrete event functions such as event scheduling, file manipulations, statistics collection, and random sample generation. The executive control program of SLAM controls the simulation by advancing time and initiating calls to the appropriate event subroutines at the proper points in simulated time. Hence, the modeler is completely relieved of the task of sequencing events to occur chronologically.

The organization for developing a discrete event model of a system using SLAM is shown in Figure 2. Basically, the SLAM user writes subroutine INTLC to establish the initial conditions for the simulation, subroutine EVENT(I) and the corresponding event routines to specify the consequence of the occurrence of EVENT(I), and subroutine OTPUT to obtain, if desired, specialized outputs of systems variables not included in the standard SLAM reports.

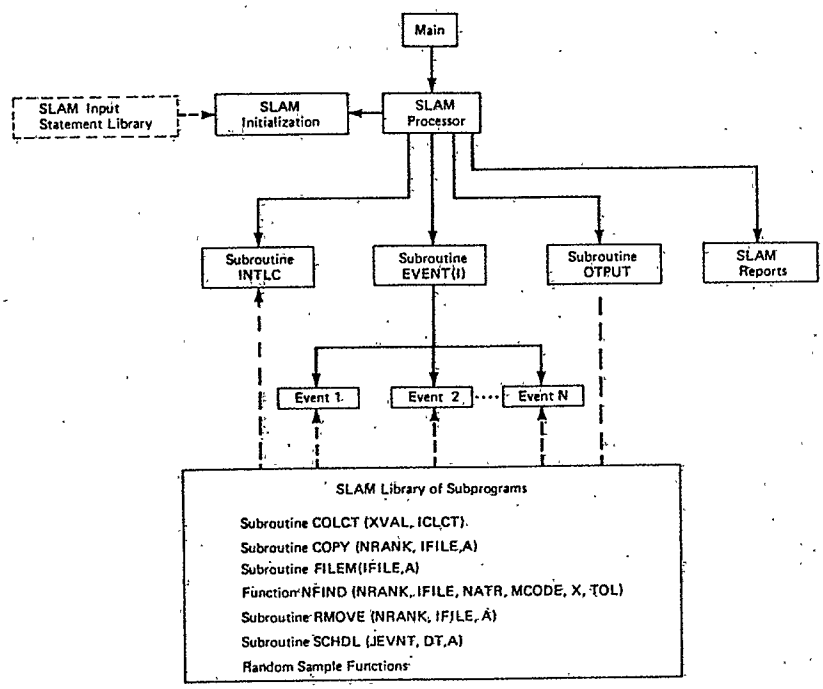


Figure 2. SLAM Organization for Discrete Event Modeling

Included in Figure 2 is a partial list of the SLAM library of subroutines for performing such functions as statistics collection (subroutine COLCT), file manipulations (subroutines COPY, FILEM, RMOVE and function NFIND), event scheduling (subroutine SCHDL), and random sample generation. From experience, this set of subprograms constitutes approximately 90 percent of the subprograms required when building a discrete event model using SLAM. SLAM does, however, contain subprograms for performing the following functions: accessing and using file entry pointers; accessing attributes of

entities; linking and unlinking entities from files; auxiliary attribute processing; report writing; accessing collected statistical data; entity tracing; statistics clearing; error reporting; table look-up and interpolation; and preparing and reporting histograms and plots.

5. AN EVENT MODEL IN SLAM

We illustrate the building of a discrete event simulation model by describing the coding of a single server queueing situation. In the coding which follows, we assume that the time between arrivals is given by the exponential distribution with a mean of 20 minutes and that the service time is uniformly distributed between 10 and 25 minutes. The operation of the system is to be simulated for a period of 480 minutes.

In this discussion, we present only the event subroutines. The logic for the arrival event, ARVL, is presented in Figure 3.

```
      SUBROUTINE ARVL
      COMMON/SCOM1/ATRIB(100),...
      EQUIVALENCE (XX(1),BUSY)
      CALL SCHDL(1,EXPON(20.,1),ATRIB)
      ATRIB(1)=TNOW
      IF(BUSY.EQ.0.) GO TO 10
      CALL FILEM(1,ATRIB)
      RETURN
   10 BUSY=1
      CALL SCHDL(2,UNFRM(10.,25.,1),ATRIB)
      RETURN
      END
```

Figure 3. Subroutine ARVL for Single Server System

The values of the SLAM discrete event variables are passed to the event routine through COMMON block SCOM1. The SLAM variable XX(1) is equivalenced to the user defined variable BUSY. The first function performed by the event is the rescheduling of the next arrival event (event code 1) to occur at the current time plus a sample from an exponential distribution with mean of 20.0 and using random stream number 1. The first attribute of the current entity is then set equal to the arrival time, TNOW. A test is then made on the variable BUSY to determine the current status of the server. If BUSY is equal to 0.0, then the server is idle and a branch is made to statement 10 where BUSY is set to 1.0 to indicate that the server is busy and the end-of-service event (event code 2) is scheduled to occur at time TNOW plus a sample from a uniform distribution between 10.0 and 25.0 using random stream number 1. Otherwise, the entity is placed in file 1 to wait for the server. In either case, the entity is identified by its arrival time which is stored as attribute 1.

The logic for the end-of-service event, ENDSV, is depicted in Figure 4.

```
      SUBROUTINE ENDSV
      COMMON/SCOM1/ATRIB(100),...
      EQUIVALENCE (XX(1),BUSY)
      TSYS=TNOW-ATRIB(1)
      CALL COLCT(TSYS,1)
      IF(NNQ(1).GT.0)  GO TO 10
      BUSY=0.
      RETURN
   10 CALL RMOVE(1,1,ATRIB)
      CALL SCHDL(2,UNFRM(10.,25.,1),ATRIB)
      RETURN
      END
```

Figure 4. Subroutine ENDSV for Single Server System

The variable TSYS is set equal to the current time, TNOW, minus the first attribute of the current entity being processed. When an event is removed from the event calendar, the ATRIB buffer array is assigned the attribute values that were associated with the event when it was scheduled. Since the value of ATRIB(1) is the entity's arrival time, the value of TSYS represents the elapsed time between the arrival and end-of-service event for this entity. A call is then made to subroutine COLCT to collect statistics on the value of TSYS as collect variable number 1. A test is made on the SLAM function NNQ(1) representing the number of entities waiting for service in file 1.

If the number of entities waiting is greater than zero, a transfer is made to statement 10 where the first entity waiting is removed from file 1 and placed onto the event calendar. The end-of-service event is scheduled to occur at time TNOW plus the service time. If no entity is waiting, the status of the server is changed to idle by setting the variable BUSY to 0.

The input statements for this example are shown in Figure 5. The GEN statement specifies the analyst's name, project title, date, and number of runs. The LIMITS statement specifies that the model employs 1 file, the maximum number of attributes is 1, and the maximum number of simultaneous entries in the system is 20. The STAT statement specifies that collect variable number 1 is to be displayed on the standard SLAM summary report with the label TIME IN SYSTEM and that a histogram is to be generated with 10 interior cells, the upper limit of the first cell is to be 0, and the cell width of each interior cell is to be 4. The TIMST statement causes time-persistent statistics to be automatically maintained on the SLAM variable XX(1) and the results to be displayed using the label UTILIZATION. The INIT statement specifies that the beginning time of the simulation is time 0 and that the ending time is time 480. The FIN statement denotes the end of all SLAM input statements. This completes the description of the discrete event model of the single server queueing situation.

```
GEN,C. D. PEGDEN,ONE SERVER,11/20/77,1;
LIMITS,1,1,20;
STAT,1,TIME IN SYSTEM,10/0/4;
TIMST,XX(1),UTILIZATION;
INIT,0,480;
FIN;
```

Figure 5. Data Statements for Single Server System

6. CONTINUOUS CAPABILITIES

A continuous model is coded in SLAM by specifying the differential or difference equations which describe the dynamic behavior of the state variables. These equations are coded by the modeler in FORTRAN by employing a set of special SLAM defined storage arrays. The value of the I^{th} state variable is maintained as variable SS(I) and the derivative of the I^{th} state variable, when required, is maintained as the variable DD(I). The immediate past values for state variable I and its derivative are maintained as SSL(I) and DDL(I), respectively. When differential equations are included in the continuous model, they are automatically integrated by SLAM to calculate the values of the state variables within an accuracy prescribed by the modeler. The event and continuous aspects of SLAM are based on GASP IV concepts (3).

Basically, a continuous model consists of a set of equations that describe the time varying performance of the system being studied. In SLAM, the equations are coded in subroutine STATE using the SS and DD variables described above. The initial values of these variables are defined through input statements or through coding included in subroutine INTLC.

SLAM computes values of the state variables in accordance with the equations specified by taking steps in time and evaluating the equations that have been coded. If difference equations have been used, SLAM takes a fixed step size. If differential equations have been employed, a variable step size using a Runge-Kutta-Fehlberg numerical integration algorithm is used. SLAM automatically determines the step size in accordance with accuracy requirements specified by the user on the CONTINUOUS input statement. The plotting of variables over time is specified through the input statements: RECORD and VAR. Changes to the parameters of the equations or to the active equations are made when state-events occur. State-events are defined in terms of state variables crossing thresholds. The request for the detection of the crossing of a threshold is specified through the SEVNT statement.

The organization for continuous modeling that is used by SLAM is shown in Figure 6.

As can be seen from Figure 6, a similarity was designed into the discrete event and continuous organizational structures to facilitate their integration for combined modeling.

7. A CONTINUOUS MODEL IN SLAM

To illustrate a model of a continuous system using SLAM, we present a model of Cedar Bog Lake that was developed by Williams (5).

The model includes three species, a solar energy supply (x_s), and the organic matter that forms a sediment on the lake bottom (x_o). These lake variables are modeled in terms of their energy content (calories/centimeter^2) and the energy transfers between the various lake variables and losses to the environment (x_e). The three species are plants (x_p), herbivores (x_h), and carnivores (x_c). The differential equations relating these species to the sediment and the solar energy source are shown on the following page.

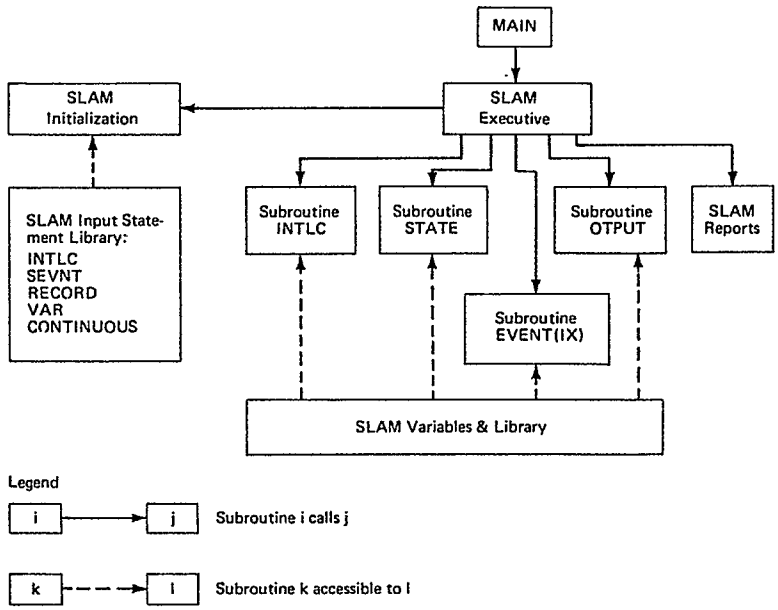


Figure 6. SLAM Organization for Continuous Modeling

$$\frac{dx_p}{dt} = x_s - 4.03x_p \ .$$

$$\frac{dx_h}{dt} = 0.48x_p - 17.87x_h \ .$$

$$\frac{dx_c}{dt} = 4.85x_h - 4.65x_c \ .$$

$$\frac{dx_o}{dt} = 2.55x_p + 6.12x_h + 1.95x_c .$$

$$\frac{dx_e}{dt} = 1.00x_p + 6.90x_h + 2.70x_c .$$

The values of the variables at time zero are: $x_p(0) = 0.83$, $x_h(0) = 0.003$, $x_c(0) = 0.0001$, $x_o(0) = 0.0$, and $x_e(0) = 0.0$.

The annual cycle in solar radiation is simulated using the following equation:

$$x_s = 95.9\ (1+0.635 \sin 2\pi t)$$

where t is time in years. These equations represent such processes as the predation of one species by another, plant photosynthesis, and the decaying of dead species. Energy transfers between lake entities and their environment are due to respiration and migration.

We will use SLAM to illustrate the procedure for obtaining the values of the variables x_p, x_h, x_c, x_o, x_c, x_s, over time. First, we make an equivalence between the model variables and the SLAM state vector SS(*) as shown on the following page.

$$SS(1) = x_p \rightarrow DD(1) = \frac{dx_p}{dt}$$

$$SS(2) = x_h \rightarrow DD(2) = \frac{dx_h}{dt}$$

$$SS(3) = x_c \rightarrow DD(3) = \frac{dx_c}{dt}$$

$$SS(4) = x_o \rightarrow DD(4) = \frac{dx_o}{dt}$$

$$SS(5) = x_c \rightarrow DD(5) = \frac{dx_c}{dt}$$

and $SS(6) = x_s$.

The entire SLAM program consists of writing the main program, subroutine STATE, and the input statements. These are shown in Figure 7. The main program is in the standard SLAM form. In subroutine STATE, the set of differential equations is coded. The translation of the equations from the model to the SLAM code is direct and normally does not require an excessive amount of work. The input statements for the model involve mainly the definitions of the variables to be plotted which is done on RECORD and VAR input statements. The CONTINUOUS statement defines the limits on the step size and the accuracy requirements for the numerical integration of the differential equations.

```
      PROGRAM MAIN(INPUT,OUTPUT,TAPE5=INPUT,TAPE6=OUTPUT,
     1TAPE7)
      DIMENSION NSET(1000)
      COMMON/SCOM1/ATRIB(100),DD(100),...
     1,NCRDR,NPRNT,NNRUN,NNSET,NTAPE,SS(100),...
      COMMON QSET(1000)
      EQUIVALENCE (NSET(1),QSET(1))
      NNSET=1000
      NCRDR=5
      NPRNT=6
      NTAPE=7
      CALL SLAM
      STOP
      END

      SUBROUTINE STATE
      COMMON/SCOM1/ATRIB(100),DD(100),...
     1,NCRDR,NPRNT,NNRUN,NNSET,NTAPE,SS(100),...
      DATA PI/3.14159/
      SS(6)=95.9*(1.+0.635*SIN(2.*PI*TNOW))
      DD(1)=SS(6)-4.03*SS(1)
      DD(2)=0.48*SS(1)-17.87*SS(2)
      DD(3)=4.85*SS(2)-4.65*SS(3)
      DD(4)=2.55*SS(1)+6.12*SS(2)+1.95*SS(3)
      DD(5)=SS(1)+6.9*SS(2)+2.7*SS(3)
      RETURN
      END

      GEN,PRITSKER,CEDAR BOG LAKE,3/5/1978,1;
      CONTINUOUS,5,1,.00025,.025,.025;
      INTLC,SS(1)=.83,SS(2)=.003,SS(3)=.0001;
      INTLC,SS(4)=0.0,SS(5)=0.0;
      RECORD,TNOW,TIME,0,P,0.025;
      VAR,SS(1),P,PLANTS;
      VAR,SS(2),H,HERBIVORES;
      VAR,SS(3),C,CARNIVORES;
      VAR,SS(4),O,ORGANIC;
      VAR,SS(5),E,ENVIRONMENT;
      VAR,SS(6),S,SOLAR ENERGY;
      INITIALIZE,0,2.0;
      FIN;
```

Figure 7. SLAM Program of Cedar Bog Lake

The INTLC statement initializes the SS(•) values as prescribed by the problem statement, and the INITIALIZE statement specifies that the simulation should start at time zero and end at time 2. This example illustrates the ease of coding continuous models in SLAM.

8. THE MANY INTERFACES OF SLAM

Interfaces have been designed into SLAM to encourage combined modeling using networks, events, and sets of equations. These interfaces consist of the following interaction capabilities:

1. An entity flowing through a network model can initiate the occurrence of a discrete event by arriving to an EVENT node. Upon the arrival of an entity, the EVENT node causes subroutine EVENT(JEVNT) to be called. The value of JEVNT specifies the event code of the discrete event to be executed. Since the logic associated with the EVENT node is coded by the modeler for use in a discrete-event model, its operational logic provides complete flexibility. Thus, a modeler faced with an operation for which no standard network node is provided can employ the event node to perform the specialized logic required.

2. The duration of activities in the network model can be specified to end when a time-event occurs, a state-event occurs, a network node is released, or by a function completely written by the modeler. The duration of an activity can be specified as STOPA(I) which means that the activity will end when the modeler calls subroutine STOPA with an argument of I from within a discrete event model. Thus, the conditions for the activity to be completed can be tested at any event time and the activity ended through a call to subroutine STOPA. When the activity is ended for the entity, the entity continues its flow through the network by arriving to the end node of the activity.

   The duration can also be specified as REL(NLBL) to indicate that the activity is to be completed when another node in the network with a node label of NLBL is released. In essence, this capability provides a network to network interface. In SLAM, a DETECT node is available for detecting the occurrence of state events, that is, a continuous variable crossing a threshold value. By assigning a label to the DETECT node, and referencing this DETECT node label in the specification, the occurrence of a state event can cause the end of an activity to occur.

   The duration can also be specified as USERF(IFN). This specification indicates that the modeler will write the FORTRAN function USERF. The duration of the activity is specified by the value given to USERF. IFN is a function number to differentiate between the use of USERF for different activities. Through the use of USERF, complex activity durations can be coded. Function USERF can also be used to assign attribute values to the entities flowing through the network.

3. An entity can be inserted into a network model from a discrete event at an ENTER node. Each ENTER node has a unique integer code NUM. An entity is inserted at the node from a discrete event when a call is made to subroutine ENTER(NUM,A), where NUM is the numeric code of the ENTER node and A contains the values of the attributes of the entity to be inserted into the network.

4. Entities in the network can cause instantaneous changes to values of state variables. This is accomplished by resetting the value of the state variable at an ASSIGN node. When an entity arrives to an ASSIGN node, the assignment specified at the ASSIGN node is made.

5. Events can cause instantaneous changes to the values of state variables through the use of statements that specify the new values for the state variables.

6. An input statement is available in SLAM for defining state events. When the conditions for the state event occur, subroutine EVENT is called with an event code that is part of the input specification. The occurrence of an event can initiate any of the changes described above. That is, other events can be scheduled, state variables can be updated, and entities can be inserted into the network through calls to subroutine ENTER. In addition, activities can be completed through calls to subroutine STOPA.

Examples of the use of each of the above interfaces have been described (1,4).

9. CHARACTERISTICS OF THE SLAM PROCESSOR

SLAM is written in standard ANSI FORTRAN. The SLAM program is completely portable and is currently running on the following computers: IBM 360/370, Univac 1108, Honeywell 6000 Series, CDC 6000 Series, Harris 550, VAX 11/780, PRIME 400 and 700 Series, MODCOMP CLASSIC, and XEROX Sigma 7.

Since 1979, over 200 copies of the SLAM program have been distributed to government, industry, and academic installations. As an example, storage requirements for the compiled version of SLAM are:

VAX 11/780: 126,000 bytes
CDC 6000 Series: 77,000 octal words, and
IBM 360/370: 128,000 bytes

10. APPLICATIONS

SLAM is a new language which was introduced in 1979. Due to the time delays for publications, there are no papers in the literature dealing with applications of SLAM. However, the distribution of the SLAM program has been extensive and we are continually receiving information about current applications.

At Pritsker and Associates, SLAM has been used to analyze proposed designs of an aerospace manufacturing facility, for assessing material handling and storage requirements at a blast furnace, to determine the effectiveness of an electronic message switching system, and to project the throughput of a new pharmaceutical production line. A few of the other applications in progress around the country are: design evaluation of multiprocessor computer systems, assessment of improvements of a group technology based plant design; and development of a training and educational program.

REFERENCES

Pegden, C. D. and A. A. B. Pritsker (1979), "SLAM: Simulation language for alternative modeling", Simulation, 33, 5, pp. 145-157.

Pritsker, A. A. B. (1979), Modeling and analysis using Q-GERT networks, second edition, Halsted Press (Division of John Wiley and Sons, Inc.), New York and Pritsker and Associates, Inc., West Lafayette, Indiana, 456 p.

Pritsker, A. A. B. (1974), The GASP IV simulation language, John Wiley and Sons, Inc., New York, 451 p.

Pritsker, A. A. B. and C. D. Pegden (1979), Introduction to simulation and SLAM, Halsted Press (Division of John Wiley and Sons, Inc.), New York and Systems Publishing Corporation, P.O. Box 2161, West Lafayette, Indiana, 588 p.

Williams, R. B. (1971), "Computer simulation of energy flow in Cedar Bog Lake, Minnesota based on the classical studies of Lindeman", in Systems analysis and simulation in ecology (B. C. Pattern, Ed.), Academic Press, New York.