Unit VI: Simulation Languages

Introduction:

Simulation languages are versatile, the general purpose of classes of simulation software that can be used to create a magnitude of modelling operation. In a sense, these languages are comparable to FORTRAN, C#, VB.NET or Java but also include specific features to facilitate the modelling process. Some examples of modern simulation languages are GPSS/H, GPSS/PC, SLX and SIMSCRIPT III. Other simulation languages such as SIMAN have been integrated into a broader development framework. Simulation language exists for discrete, continuous and agent-based modelling paradigm.
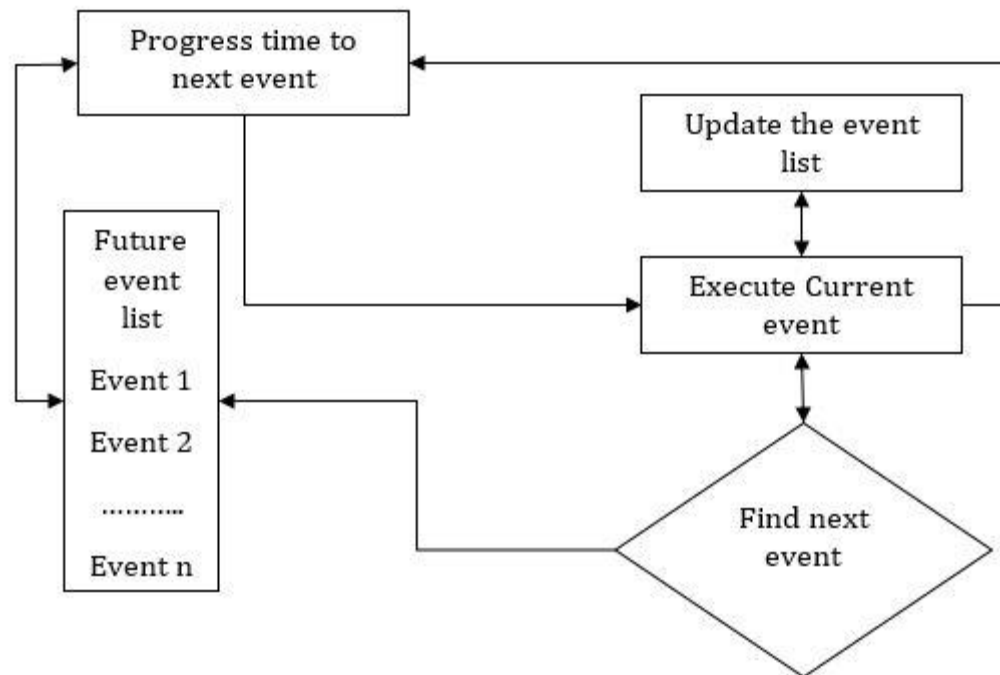

Features:

Specialized features usually differentiate from a general programming language. These features are intended to free the analyst from recreating software tools and procedures used by virtually all modelling application. Not only the development of these features be time-consuming and difficult, but without them, the consistency of a model could vary and additional debugging, validation and verification would be required. Most simulation languages should have the following features:

1. Simulation clock or a mechanism for advancing simulated time.
2. Methods to schedule the accuracy of the event.
3. Tools to collect and analyse statistics concerning the uses of various resources and entities.
4. Tools for reporting the result.
5. Debugging and error detection facilities.
6. Random number generator and related set of tools.
7. The General framework for model creation.

Working on Simulation Language:

Most discrete event simulation language model a system by updating the simulation clock to the time that the next event is scheduled to occur. Events and their scheduled times of occurrence are maintained automatically on one of two order list, the current event chain or future event chain. The current event chain keeps a list of all events that will occur at the present clock time. The future event chain is a record of all events that can occur at some point in the future. A simulation clock moves to the next event on the future events chain and changes the system state of the model based on the characteristics of that event.

Merits of Simulation Language:

1. Since most of the features to be programmed are built-in simulation language takes comparatively less programming time and effort.

2. Since simulation language consists of blocks, specially constructed to simulate the common feature, they provide a natural framework for simulation and modelling.

3. Simulation models coded in simulation language can be easily changed and modified.

4. The error detection and analysis are done automatically in a simulation language.

5. Simulation models developed in simulation language, especially the specific application package called simulators are very easy to use.

Features of Simulation Software:

1. Modelling Flexibility:

The simulation must be flexible enough to adapt to change the configuration. The domain of application should be wide and the model should be equally valid over the whole domain.

2. Ease of Modelling:

It should be easy to develop the simulation model, easy to debug the program and validate the model. The software should have an in-built interactive debugger, error detector and online health.

3. Fast Execution Speed:

The speed of execution is always a very important requirement of any simulation model. It is especially an essential feature in case of simulation of a large and complex system.

4. Compatibility to Various Computer System:

The simulation language should be compatible with various computer system like microcomputer, engineering workstation minicomputers and mainframe computers.

5. Statistical Capabilities:

A simulation software should contain multiple stream random number generators and as many standards probability distributions as possible. It should have blocks for designing the statistics like the simulation run, warming up period and the number and the length of replication.

6. The Capability of Animation:

The animation capability of the simulation package is one of the main reasons for the popularity of simulation language. The animation is carried out in two modes i.e. concurrent mode and playback mode.

## 7. Report Presentation Capabilities:

The output of the simulation the model should be well documented and illustrated. The user should be able to get the information of its interest in the easily understandable form.

Types of Simulation Languages:

A variety of simulation languages exist and are used by business, researchers, manufacturing and service companies and consultant. The two common simulation languages are GPSS and SIMSCRIPT.

1. GPSS (General Purpose Simulation System):

GPSS was originally developed by Geoffrey Gordon of IBM and released in October 1961. Following IBM release of GPSS to the public domain it became a multi-vendor simulation language and has been in continuous use since.
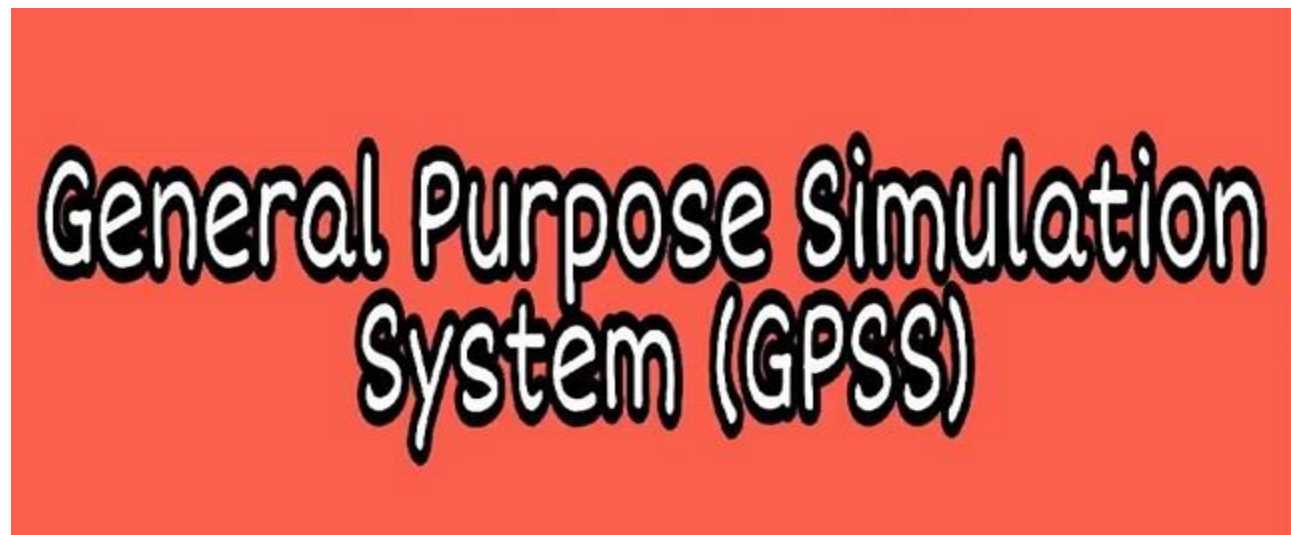
In general, GPSS enjoys widespread popularity due to its sensible world view and power. Its basic function can be easily learned while powerful features make it ideal for modelling complex systems. It is used to simulate the queuing system that consists of customer entities interacting and completing a system of constraint resources. The resources are structured as a network of blocks that entities are entered and used to perform various task in the sudden amount of simulated time. As entities move through these networks, which have been organized to represent a real-world system, statistics are collected and used to determine if the system contains bottleneck, is over or underutilization or exits other characteristics. Output data is made available for the analyst at the end of the production run.

Discrete Systems Modelling and Simulation with GPSS:

To represent the state of the system in the model, a set of numbers is used that is used in the discrete-time system. A number, state descriptor is used to represent some aspect of the system state. Some state descriptors range over values that have physical significance and some represent conditions.

The effect of the state descriptors is to change the value as the simulation proceeds. We define a discrete event as a set of circumstances that causes an instantaneous change in one or more system state descriptors. It is possible that two different events occur simultaneously, or are modelled as being simultaneous so that not all changes of state descriptors occurring simultaneously necessarily belong to a single event.

The term simultaneous refers to the occurrence of the changes in the system, and not to when the changes are made in the model in, which, of necessity, the changes must be made sequentially. A system simulation must contain a number representing time. The simulation proceeds by executing all the changes to the system descriptors associated with each event, as the events occur, in chronological order. How events are selected for execution (when there are simultaneous events) is an important aspect of programming simulations.

For writing discrete system simulation programs, several programming languages have been produced. These programs embody a language with which to describe the system and a programming system that will establish a system image and execute a simulation algorithm. Each language is based upon a set of concepts used for describing the system. The term

worldview has come to be used to describe this aspect of simulation programs. The user of the program must learn the world-view of the particular language he is using and to describe the system in those terms. Given such a description, the simulation programming system can establish a data structure that forms the system image. It will also compile and sometimes supply the routines to carry out the activities. One of the most commonly used simulation languages is GPSS. It illustrates the divergence in design consideration. GPSS has been written especially for the user with little or no knowledge of programming experience. The simplification of GPSS results in some loss of flexibility. GPSS applies to a wide variety of systems.

GPSS Programs:

The General-Purpose Simulation System language has been developed over many years, principally by the IBM Corporation. It has been implemented on several different manufacturers' machines and there are variations in the different implementations. GPSS V is more powerful and has more language statements and facilities than GPSS/360.The GPSS V is implemented by IBM Corporation.

General Description:

The system to be simulated in GPSS is described as a block diagram in which the blocks represent the activities and lines joining the blocks indicate the sequence in which the activities can be executed. Where there is a choice of activities, more than one line leaves a block and the condition for the choice is stated at the block.

To base a programming language on this descriptive method, each block must be given a precise meaning. The approach taken in GPSS is to define a set of 48 specific block types, each of which represents a characteristic action of the systems. Only the specified block types are allowed while drawing the block diagram of the system. Each block type is given a name that is descriptive of the block action and is represented by a particular symbol. Each block type has several data fields.
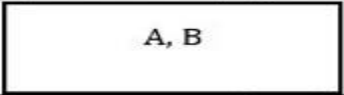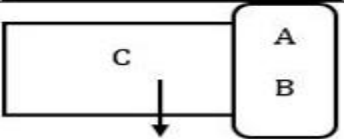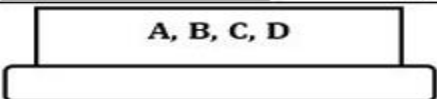
Moving through the system being simulated are entities that depend upon the nature of the system. In a communication system, the entities of concern are messages, which are moving. Meanwhile, in a road transportation system, the entities that are moving are the motor vehicles. A data processing system is concerned with records. In the simulation, these
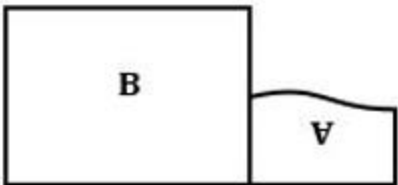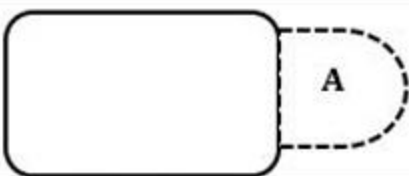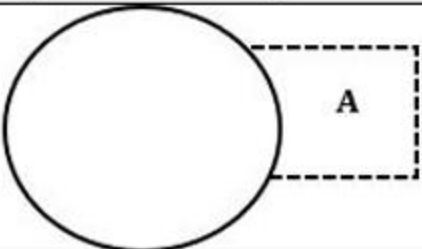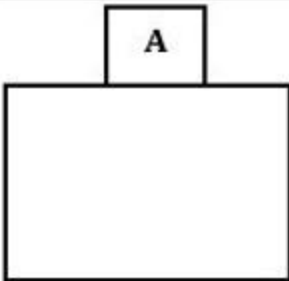
entities are called transactions. The sequence of events in real-time is reflected in the movement of transactions from block to block in simulated time.
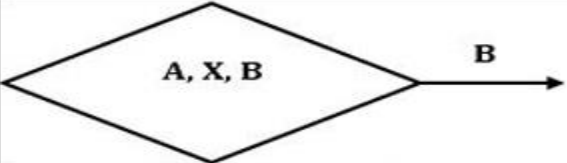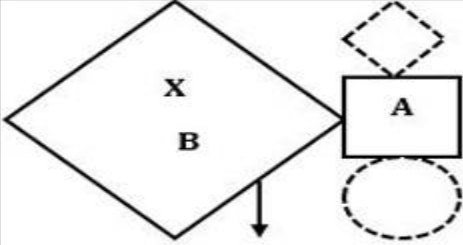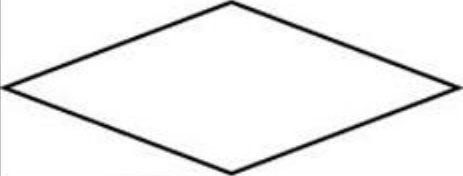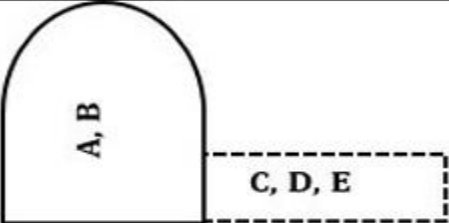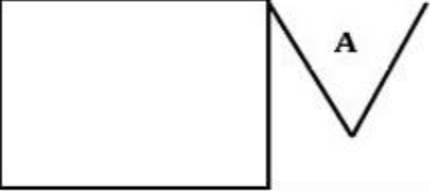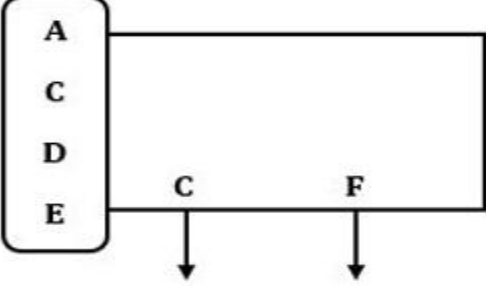
Transactions are created at one or more GENERATE blocks and are removed from the simulation at TERMINATE blocks. There can be many transactions simultaneously moving through the block diagram. Each transaction is always positioned at a block and most blocks can hold many transactions simultaneously. The transfer of a transaction from one block to another occurs instantaneously at a specific time or when some change of system condition occurs.
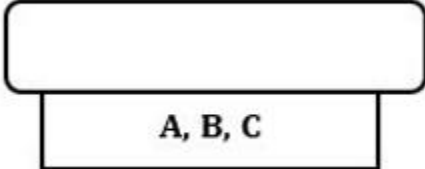
A GPSS block diagram can consist of many blocks up to some limit prescribed by the program (usually set to 1000). An identification number called a location is given to each block, and the movement of transactions is usually from one block to the block with the next highest location. The locations are assigned automatically by an assembly program within GPSS so that, when a problem is coded, the blocks are listed in sequential order. Blocks that need to be identified in the programming of problems are given a symbolic name. The assembly program will associate the name with the appropriate location. Symbolic names of blocks and other entities of the program must be from three to five non-blank characters of which the first three must be letters.

GPSS Block Diagram Symbols:

S.N.	Representation/Symbols	Meaning
1	A, B	Advance
2	C A B	Link
3	A	Seize
4	A, B, C, D	Assign

5		Logic
6		Tabulate
7		Depart
8		Mark
9		Terminate
10		Enter
11		Priority

12		Test
13		Gate

14		Queue
15		Transfer
16		Generate
17		Release
18		Unlink

19	A	Leave
20	A, B, C	Save Value

Basic Concepts:

Action times:

Clock Time is represented by tan integral number, with the interval of real-time corresponding to a unit of time chosen by the program user. The unit of time is not specifically stated but is implied by giving all the time in terms of the same unit. One block type called ADVANCE is concerned with representing the expenditure of time. The program computes an interval of time called action time for each transaction as it enters an ADVANCE block and the transaction remains at the block for this interval of a simulated time before attempting to proceed. The only other block type that employs action time is the GENERATE block, which creates transactions. The action time at the GENERATE block controls the interval between successive arrivals of transactions.

The action time may be a fixed interval or a random variable, and it can be made to depend upon conditions in the system in various ways. Action time is defined by giving a mean and modifier as the A and B fields for the block. If the modifier is zero, the action time is a constant equal to the mean. If the modifier is a positive number (<=mean), the action time is an integer random variable chosen from the range (mean ± modifier), with equal probabilities given to each number in the range.

By specifying the modifier at an ADVANCE or GENERATE block to be a function, the value of the function controls the action time. The action time is derived by multiplying the mean by the value of the function. Various types of input can be used for the functions, allowing the functions to introduce a variety of relationships among the variable of a system. In particular,

by making the function an inverse cumulative probability distributed, and using as input a random number which is uniformly distributed, the function can provide a stochastic variable with a particular non-uniform distribution.

The GENERATE block normally begins creating transactions from zero time and continues to generate them throughout the simulation. The C field, however, can be used to specify an an-offset time as the time when the first transaction will arrive. If the D field is used, it specifies a limit to the total number of transactions that will come from the block. Transactions have a priority level and they carry items of data called parameters. The E field determines the priority of the transactions at the time of creation. If it is not used, the priority is of the lowest level.

Parameters can exist in four formats. They can be signed integers of the full word, halfword, or byte size, or they can be signed floating-point numbers. If no specific assignment of parameter type is made, the program creates transactions with 12 halfword parameters. The C, D and E fields, also, will not be needed.

The Succession of Events:

The program maintains records of when each transaction in the system is due to move. It proceeds by completing all movements that are scheduled for execution at a particular instant of time and that can logically be performed. Where there is more than one transaction due to move, the program processes transactions in the order of their priority class, and on a first-come, first-served basis within a priority class.

Once the program has begun moving a transaction it continues to move the transaction through the block diagram until one of the several circumstances arises.

The transaction may enter an ADVANCE block with a non-zero action time, in which case, the program will turn its attention to other transactions in the system and return to that transaction when the action time has been expended.

The conditions in the systems may be such that the action the transaction is attempting to execute by entering a block cannot be performed at the current time. The transaction is said to be blocked and it remains at the block it last entered. The program will automatically detect when the blocking condition has been removed and will start to move the transaction again at that time.

A third possibility is that the transaction enters a TERMINATE block, in which case it is moved from the simulation. A fourth possibility is that a transaction may be put on a chain.

When the program has moved one transaction as far as it can go, it turns its attention to any other transactions due to move at the same time instant. If all such movements are complete, the program advances the clock to the time of the next most imminent event and repeats the process of executing events.

Choice of Paths:

The TRANSFER block allows some location other than the next sequential location to be selected. The choice is normally made between two blocks referred to as next blocks A and B. The method used for choosing is indicated by a selection factor in the field A of the TRANSFER block. It can be set to indicate one of the nine choices. Next blocks A and B are placed in fields B and C, respectively. If no choice is to be made, the selection factor is left blank. An unconditional transfer is then made to next block A.

A random choice can be made by setting the selection factor, S, to a three-digit decimal fraction. The probability of going to next block A is then 1-S, and to the next block B it is S. a conditional mode, indicated by setting field A to BOTH, allows a transaction to select an alternate path depending upon existing conditions. The transaction goes to next block A if this move is possible, and to the next block B if it is not. If both moves are impossible the transaction waits for the first to become possible, giving preference to A in the event of simultaneity.

GPSS Programs Application:

Let us consider a scenario representing a machine tool in a manufacturing shop which is turning out parts at the rate of every 5 minutes. The parts are then examined by the inspector who takes 4 3 minutes for each part and rejects about 10% of the parts. We can represent each part by one transaction, and the time unit selected for the problem will be 1 minute. A block diagram representing the system is given below:
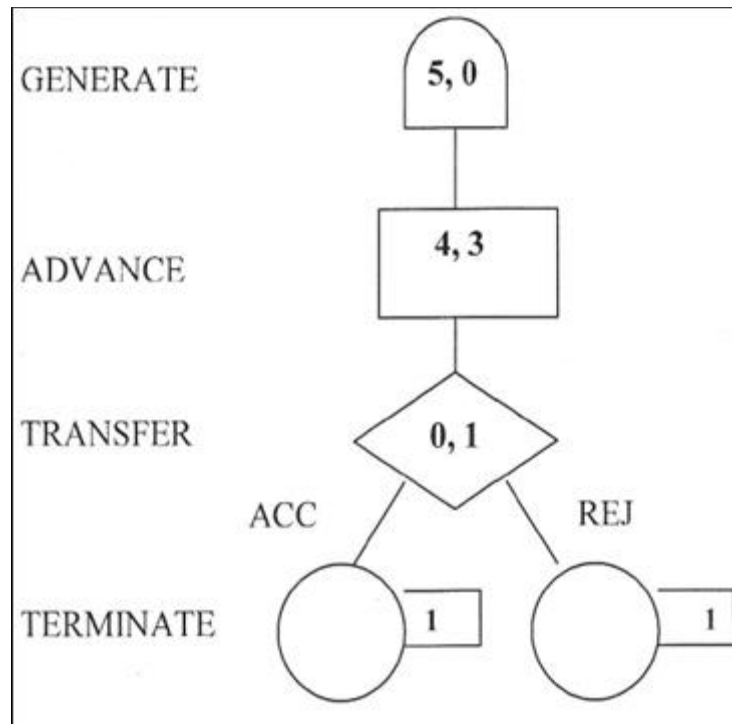
Fig: Manufacturing Shop – Model 1

In the block diagram, the block location is placed at the top of the block, the action time is indicated in the centre in the form T = a, b where a is the mean MD b is the modifier and the selection factor is placed at the bottom of the block.

A GENERATE block is used to represent the output of the machine by creating one transaction every five units of time. An ADVANCE block with a mean of 4 and modifier of 3 is used to represent an inspection. The time spent on inspection will, therefore, be any one of the values 1,2, 3, 4, 5, 6 or 7, with equal probability given to each value. Upon completion of the inspection, transactions go to a TRANSFER block with a selection factor of 0.1, so that 90% of the parts go to the next location called ACC, to represent accepted parts and l0% go to another location called REJ to represent rejects. Both locations reached from the TRANSFER block are TERMINATE blocks.

The problem can be coded. Column 1 is only used for a comment card. A field from columns 2 to 6 contains the location of the block where it must be specified. The GPSS program will automatically assign sequential location numbers as it reads the statements, so it is not usually necessary for the user to assign locations. The TRANSFER block, however, will need

to make reference to the TERMINATE blocks to which it sends transactions, so these blocks have been given symbolic location names ACC and REJ.

The second section of the coding, from columns 8 to 18, contains the block type name, which must begin in column 8. Beginning in column 19, a series of fields may be present, each separated by commas and having no embedded blanks. Anything following the first blank is treated as a comment. The meaning of the fields depends upon the block type.

The program also accepts input in a free format, in which the location field, if used, begins in column 1; but none of the other fields has a fixed starting position. Instead a single blank mark the transition from the location field to the operation field and from the operation held to the operands. If no location is specified, an initial blank is used.

For the TRANSFER block, the first field is the selection factor, and the B and C fields are exits 1 and2, respectively. In this case, exit 1 is the next sequential block, and it would be permissible to omit the name ACC in both the TRANSFER field B and the location field of the first TERMINATE block. It should also be noted that when a TRANSFER block is used in an unconditional mode, so that field A is not specified, a comma must still be included to indicate the field.

The program runs until a certain the count is reached as a result of transactions terminating. Field A of the TERMINATE block carries a number indicating by how much the termination count should be incremented when a transaction terminates at that block. The number must be positive and it can be zero, but there must be at least one TERMINATE block that has a non-zero field A. in this case, both TERMINATE blocks have 1; so the terminating counter will add up the total number of transactions that terminate, in other words, the total number of both good and bad parts inspected.

A control statement called START indicates the end of the problem definition and contains, in field A, the value the terminating counter is to reach to end the simulation. In this case, the START Statement is set to stop the simulation at a count of 1,000. Upon reading a START statement, the program begins executing the simulation.

When the simulation is completed, the program automatically prints an output report, in a prearranged format, unless it has been instructed otherwise.

The problem input is printed first, with the locations assigned by the problem listed to the left, and a sequential statement number on the right. The first line of the output following the listings gives the time at which the simulation stopped. The time is followed by a listing of block counts. Two numbers are shown for each block of the model. On the left is a count of how many transactions were in the block at the time the simulation stopped, and on the right is a figure showing the total number of transactions that entered the block during the simulation.

Facilities and storage:

Associated with the system being simulated are many permanent entities, such as items of equipment, which operate on the transactions. Two types of permanent entities are defined in GPSS to represent system equipment.

A facility is defined as an entity that can be engaged by a single transaction at a time. Storage is defined as an entity that can be occupied by many transactions at a time, up to some predetermined limit. A transaction controlling a facility, however, can be interrupted or preempted by another transaction. Also, both facilities and storages can be made unavailable, as would occur if the equipment they represent breaks down, and can be made available again, as occurs when a repair has been made.

There can be many instances of each type of entity to a limit set by the program (usually 300), Individual entities are identified by number, a separate number sequence being used for each type. The number 0 for these and all other GPSS entities is illegal.

Block types SEIZE, RELEASE, ENTER and LEAVE are concerned with using facilities and storages. Field A in each case indicates which facility or storage is intended, and the choice is usually marked in the flag attached to the symbols of the blocks. The SEIZE block allows a transaction to engage a facility if it is available. The RELEASE block allows a transaction to disengage the facility. An ENTER block allows a transaction to occupy a pace in storage if it is available, and the LEAVE block allows it to give up space. If the fields B of the ENTER and LEAVE blocks are blank, the storage contents are changed by 1. If there is a number (>=1), then the contents change by that value. Any number of blocks may be placed between the points at which a facility is seized and released to simulate the actions that would be taken while the transaction has control of a facility. Similar arrangements apply for making use of storages.

To illustrate the use of these block types, consider again the manufacturing shop. Since the average inspection time is 4 minutes and the average generation rate is one every 5 minutes, there will normally be only one part inspected at a time. Occasionally, however, a new part can arrive before the previous part has completed its inspection. This situation will result in more than one transaction being at the ADVANCE block at one time.

Assuming that there is only one inspector, it is necessary to represent the inspector by a facility, to simulate the fact that only one part at a time can be inspected. A SEIZE block and a RELEASE block needs to be added to simulate the engaging and disengaging of the inspector.
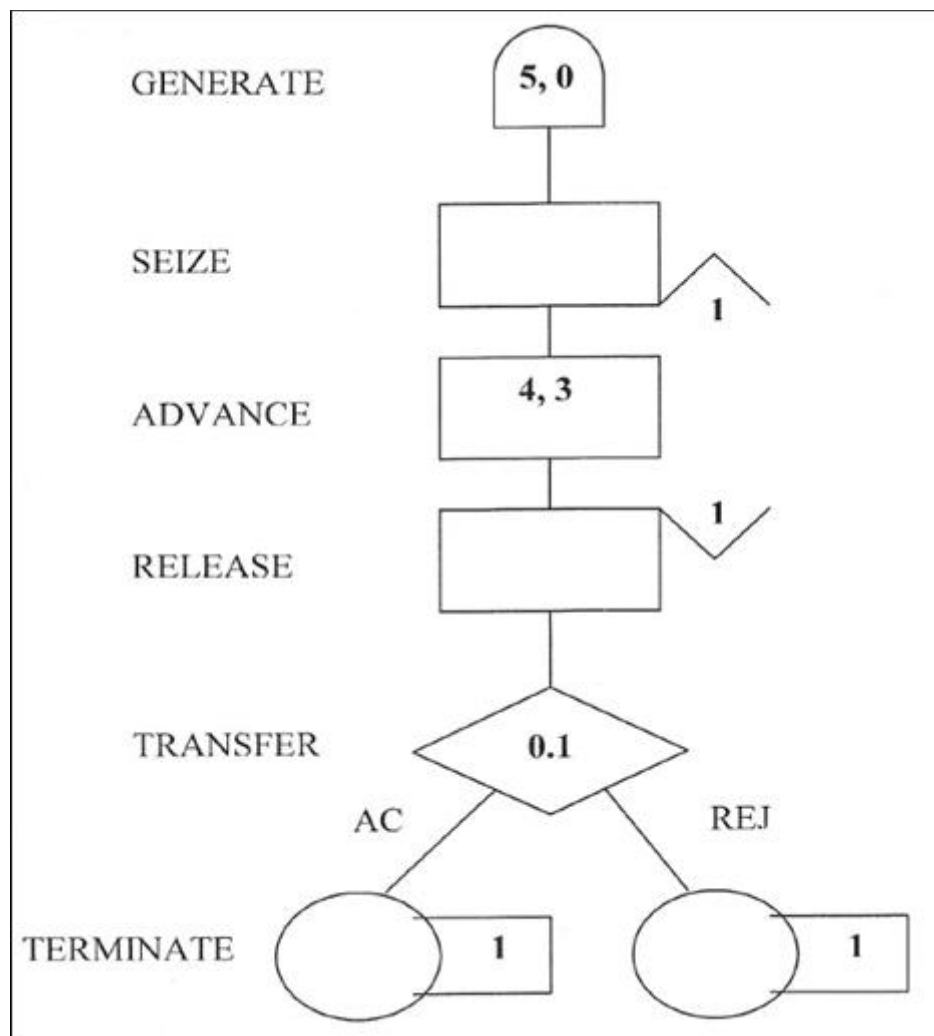

Fig: Manufacturing Shop – Model 2

If more than one inspector is available, they can be represented as a group by storage with a capacity equal to the number of inspectors. The SEIZE and RELEASE blocks of the previous

model are then replaced by an ENTER and a LEAVE block, respectively. Suppose, for example, the inspection time were three times as long as before; three inspectors would be justified.

The case of a single inspector can be modelled by using storage with capacity 1 instead of the facility. An important logical difference between the two possible representations is that, for a facility, only the transaction that seized the facility can release it, whereas, for storage, entering and leaving can be separate actions carried out independently by different transactions.

2. SIMSCRIPT III:

This language is a direct descendant of the original SIMSCRIPT language produced at Rand Corporation in 1960's. SIMSCRIPT III has constructs that allow a modular to approach a problem from either a process or an event-oriented world view. It offers unique features which are:

a. Object Oriented Programming

b. Modularity

c. SIMSCRIPT III development studio

d. Object Oriented SIMSCRIPT III graphics

e. Database Connectivity (SDBC)

In general, SIMSCRIPT III is a free form of language with English like a statement. This syntax allows the code in the system to become self-documenting. Model components can be

programmed clearly enough to provide an excellent representation of the organization and logic of the system being simulated.

SIMSCRIPT Program:

SIMSCRIPT is a very widely used language for simulating the discrete system. The language can be considered as more than just a simulation language since it can be applied to general programming problems. The description of the language given is organized in five levels.

1. Beginning with simple teaching, level to introduce the concept of programming.
2. The description add level corresponding to a specific programming language.
3. A general-purpose language.
4. A list processing language, needed for creating the data structure of a simulation model.
5. The simulation-oriented function needed to control the simulation and gathering statistics.

SIMSCRIPT System Concept:

The system to be simulated is considered to consist of entities having attributes that interact with activities. The interaction causes events that change the state of the system. In the system, SIMSCRIPT uses the term entities and attributes. To make the program efficiency it must distinguish between temporary and permanent entities and attributes.

The temporary entities represent those entities are created and destroyed during the execution of a simulation while permanent entities represent that entities remain during the run.

A special emphasis is placed on how temporary entities form sets. The user can define sets and facilities are provided for entering and removing entities into and from sets.

Activities are considered as extending overtime with their beginning and being marked as events occurring instantaneously. Each type of event is described by an event routine, each of which is given a name and a program as a separate closed routine.

A distinction is made between the indigenous (internal) event, which arises from action within the system and the exogenous (external) event, which arises from the action in the system environment.

An indigenous event is caused by a scheduling statement in some event routine. The event marking the beginning of activity will usually schedule the event that marks the end of the activity. If the beginning or end of activity implies the beginning of some activity, then the event routine marking the beginning or end of the first activity will schedule the beginning of the second.

An exogenous event requires the reading of data supplied by the user. Among the data is time at which, the event occurs. There can be many data sets representing a different set of external events. Events routines are needed to execute the changes that result when an external event becomes due for execution. Part of the automation initialization procedure of SIMSCRIPT is to prepare the first exogenous event for each data set.

In practice, external event, such as arrivals are often generated using the bootstrap method. Given the statistical distribution of the inter-arrival time, an indigenous event routine continuously creates one arrival from its predecessor. An exogenous event routine is essential only when specific data are needed.

Organization of SIMSCRIPT Program:

Since the event routines are closed routines, some means must be provided for control between them. The transfer is affected by the use of event notice which is created when it is determined that an event is scheduled. At all time, an event notice exists for every indigenous event scheduled to occur either at current clock time or in the future event.

Each event notice records the time is due to occur and the event routine that is to execute the event. If the event is to involve one of the temporary entities, the event notice will usually identify which one is involved.

The event notice is filed in chronological order. When all events that can be executed at a particular time have been processed, the clock is updated to the time of the next event notice and control is passed to the event routine identified by the notice. Their actions are automatic and do not need to be programmed and event notice do not usually go to more than one activity.

If the event executed by a routine result in another event, either at the current clock time or in future, the routine must create a new event notice and file it with the other notice.

In the case of the exogenous event, a series of exogenous event statement are created one for each event. All exogenous event statements are sorted to chronological order and they are read by the programs when the time for the event is due to occur.
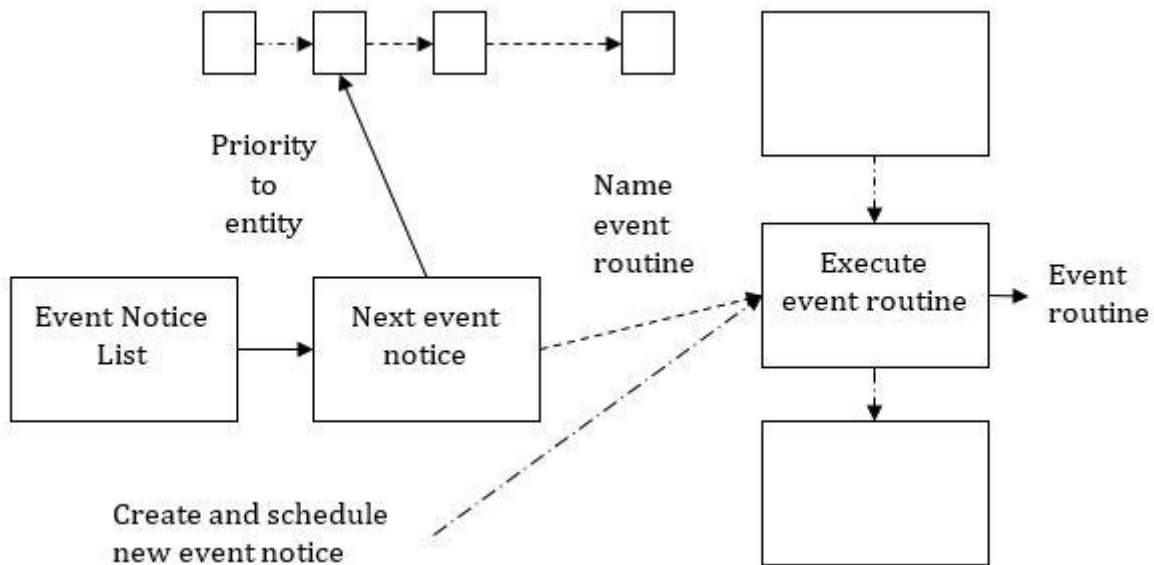

Fig: Organization of SIMSCRIPT Program