Simulation and Modeling
Simulation Languages

Er. Narayan Sapkota, M.Sc.
narayan.sapkota@eemc.edu.np

Everest Engineering College (Senior Lecturer)
Kathmandu University (Visiting Faculty)

January 5, 2025

Syllabus

1. Types of Simulation Languages
2. Discrete Systems Modeling and Simulation with GPSS
3. GPSS Programs Applications
4. SIMSCRIPT -Organization of A SIMSCRIPT Program
5. SIMSCRIPT Programs

Table of Contents

Types of Simulation Languages (1)

Continuous System Simulation

Continuous system simulationinvolves modeling systems where state variables change continuously over time. These systems can be represented by differential equations that describe the relationship between various components or states of the system. CSSLs are specifically designed to handle such systems.

- Nature of the System: Continuous systems model processes where variables change smoothly over time. Examples include fluid flow in pipes, electrical circuits, or population dynamics.
- Mathematical Representation: The models often involve differential equations or integral equations that describe how the system's state evolves over time.
- Execution: In continuous simulation, the state of the system is updated continuously and instantaneously based on the equations governing the system's dynamics.

Types of Simulation Languages (2)

- Examples of CSSLs:
  - SIMULINK: A widely used software tool within MATLAB that helps in modeling and simulating continuous-time systems, such as control systems, signal processing, and other dynamic systems.
  - DYNAMO: A language used for modeling continuous systems in fields like ecology, economics, and engineering.
  - VISSIM: While mainly for traffic simulation, it uses continuous methods to model vehicle flow and congestion.

Discrete System Simulation Language (DSSL)

Discrete system simulation is used for modeling systems where changes occur at discrete points in time, often in response to specific events. These changes are typically represented by a sequence of events, rather than continuous changes in state. Discrete systems are generally event-driven, where each event occurs at a specific time and leads to a change in the state of the system.

Types of Simulation Languages (3)

- Nature of the System: Discrete systems are modeled as sequences of distinct events occurring at distinct points in time. Each event causes a change in the system's state, and the system evolves step by step. Examples include queueing systems, computer networks, manufacturing systems, or inventory systems.
- Mathematical Representation: The models typically use difference equations, event-based logic, or other methods to represent state changes at discrete time steps.
- Execution: In discrete simulation, the system state is updated at specific intervals or events, often by a "clock" that advances when an event occurs.
- Examples of DSSLs:
    - SIMSCRIPT: A popular language used for discrete event simulation, especially in operations research, logistics, and industrial engineering.

Types of Simulation Languages (4)

- GPSS (General Purpose Simulation System): Another prominent DSSL used for simulating systems such as manufacturing, transportation, and telecommunication networks.
- Arena: A visual simulation language that allows for modeling discrete systems like production lines, supply chains, and service systems.
- AnyLogic: A powerful tool for simulating both discrete and continuous systems, but with a focus on discrete event modeling.

Types of Simulation Languages (5)

Aspect	CSSL	DSSL
Type of System	Continuous (state variables change continuously)	Discrete (state changes occur at specific events)
Mathematical Representation	Differential equations and integral equations	Event-driven or difference equations
Execution Model	Updates the system state continuously over time	Updates the system state at discrete events/times
Examples	SIMULINK, DYNAMO, VISSIM	SIMSCRIPT, GPSS, Arena, AnyLogic
Applications	Physical systems (circuits, fluid dynamics), biological processes	Queuing, manufacturing, inventory systems, networks

Types of Simulation Languages (6)

Hybrid Simulation Languages

- These combine both discrete and continuous simulation methods, enabling the modeling of systems with both discrete events and continuous processes.
- Dynamo: A modeling language for system dynamics, typically used in business, economics, and environmental modeling.
- Vensim: A system dynamics simulation tool, often used for policy analysis, decision-making, and dynamic system modeling.

Agent-Based Simulation Languages

- These focus on modeling systems in which individual entities (agents) with defined behaviors interact with each other. These are useful for simulating social behaviors, ecology, and market dynamics.

Types of Simulation Languages (7)

- NetLogo: A popular language for agent-based modeling, used widely in education and research.
- Repast: A widely-used suite of agent-based modeling tools that allows for large-scale agent-based simulations.

Mathematical and Statistical Simulation Languages

- These languages focus on simulations using mathematical equations, statistical methods, and data analysis techniques.
- MATLAB: A high-level language often used for numerical computing, modeling, and simulations.
- R: A statistical programming language that can be used for simulations, particularly in the context of statistical models and data analysis.

Table of Contents

Discrete Systems Modeling and Simulation with GPSS (1)

To represent the state of the system in the model, a set of numbers is used that is used in discrete time system.

State Descriptors

- It represent various aspects of a system's state. These descriptors are numerical values that may have physical meanings (e.g., the number of customers in a queue or the availability of a machine) or represent conditions (e.g., system failure, idle state).
- These descriptors change as the simulation progresses, and their values are updated when events occur.
- Example: State Descriptors in a queueing system might include the number of customers in the queue (an integer value) and the status of the server (either busy or idle).

Discrete Systems Modeling and Simulation with GPSS (2)

Discrete Events

- A discrete event is a specific occurrence that causes an instantaneous change in one or more state descriptors.
- Events are often modeled as a single moment in time where something significant happens, such as a customer arriving at a bank or a machine breaking down.
- It is important to note that simultaneous events can occur, where multiple events may happen at the same point in time, even though they are executed sequentially in the simulation.

Event Handling and Chronological Order

- The simulation progresses by processing events in chronological order, meaning events are executed as they occur in time.

Discrete Systems Modeling and Simulation with GPSS (3)

- Even if two or more events occur simultaneously, the system handles each event sequentially, one at a time.
- The selection of events for execution becomes crucial, especially when multiple events occur at the same time, requiring careful scheduling to ensure the simulation's accuracy.

Simulation Programming Languages

- Simulation programming languages are designed to help describe a system and its dynamics in a formal way. These languages allow the user to define how the system behaves and interact with its components (state descriptors, events, etc.).
- World-View refers to the set of concepts or the framework a simulation language uses to describe a system. Each simulation language has its own world-view, and users must understand this framework to model their systems effectively.

Table of Contents

GPSS Introduction (1)

General Purpose Simulation System (GPSS) is a simulation language used for discrete-event simulations. It is especially useful in the modeling of queuing systems, with many statistics being collected automatically. The typical simulation consists of transactions being generated in the system (usually at a certain interval), performing a defined set of rules (like using a resource, waiting, transferring), and being removed from the simulation.

GPSS was developed in the 1960s by Geoffrey Gordon, an employee of IBM's Advanced Systems Development Division (ASDD). This division was heavily involved with research into the design of teleprocessing systems, trying to achieve an economic balance of the use of computer resources and shared lines between server terminals. The simulation system, then known as the Gordon Simulator, became very popular in the study of teleprocessing systems within ASDD. It subsequently was fixed and documented on October

GPSS Introduction (2)

25, 1960, in an internal IBM memorandum.

Between the winter and summer of 1961, a group of three programmers (including Gordon) rewrote the simulation system with a new algorithm and new block types. It was officially released as a supported IBM-label program on September 27, 1961, with 25 block types. At this point, Gordon stopped working on the simulation system.

- In 1963, GPSS II was released with 32 block types. It introduced system numerical attributes, which allowed tracking the current content of a Storage, the length of a Queue, or the current clock time.
- In 1965, GPSS III was released. It was made available for IBM Systems 7090/94 and 7040/44.
- In 1967, GPSS/360 was released to run on the newly released System 360.

GPSS Introduction (3)

- In 1970, GPSS V was released with 48 block types.
- In the 1980s, GPSS/VAC and GPSS/PC were released. These appear to be the last official IBM-label releases before the language became unlicensed.
- Over time, other implementations were developed for systems including large-scale Univac systems, DEC's VAX, Univac 1108, and CDC.
- In 2001, a graphical Windows program called *GPSS World* was released with new features to GPSS. It includes scripting with PLUS (a Pascal-like language), graphical system state displays, graphing, and optimization experiments.
- In 2009, a Java-based tool called *JGPSS* (Java General Purpose Simulation System) was developed to teach the GPSS simulation language.

GPSS: General Description (1)

- 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 sequences in which the activities can be executed.
- To base a programming language on an 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 systems.
- 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 a number of data fields. Figure 1 and 2 shows the symbols used for the block types.

GPSS: General Description (2)

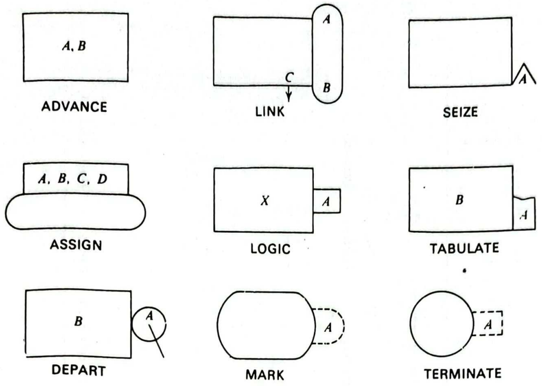


Figure 1: GPSS Block-Diagram Symbols

GPSS: General Description (3)

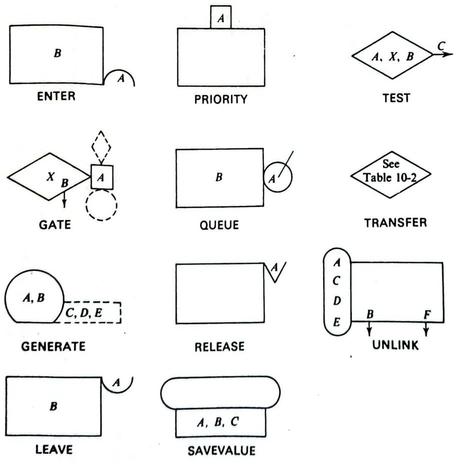


Figure 2: GPSS Block-Diagram Symbols

GPSS: General Description (4)

- In the simulation, entities that represent aspects of the system–such as messages in communication systems, vehicles in transportation systems, or records in data processing systems–are called transactions. The real-time sequence of events is mirrored by the movement of these transactions from one block to another in simulated time.
- Transactions are created at one or more GENERATE blocks and are removed at TERMINATE blocks in the simulation. Multiple transactions can move through the system simultaneously. Each transaction is always located at a specific block, and most blocks can accommodate several transactions at once. The movement of a transaction from one block to the next happens immediately, either at a set time or when a change in the system's state occurs.

GPSS: General Description (5)

- A GPSS block diagram can include up to 1,000 blocks, with each block assigned a unique identification number called a location. Transactions typically move from one block to the next in ascending order of location. These locations are automatically assigned by an assembly program within GPSS, ensuring the blocks are listed sequentially when the system is coded. Blocks that require identification for programming purposes are given symbolic names. These names must consist of 3 to 5 characters, with the first three being letters. The assembly program links these symbolic names to the corresponding block locations.

GPSS: Action Times (1)

- Clock time in GPSS is represented by an integer, where the program user defines the unit of time. This unit is not explicitly stated but is consistent throughout the simulation. The ADVANCE block is used to represent the passage of time. When a transaction enters this block, the program calculates how long it will stay, called the action time, and the transaction stays in the block for that amount of simulated time before moving on.
- The GENERATE block also uses action time to control how often transactions arrive. The action time can be a fixed time or vary randomly, based on system conditions. It is defined by a mean and a modifier. If the modifier is zero, the action time is fixed at the mean value. If the modifier is positive, the action time becomes a random value within a range defined by the mean and modifier.

GPSS: Action Times (2)

- The action time can also depend on a function, which can use inputs like random numbers to introduce different relationships or randomness into the simulation. For example, it can model a non-uniform distribution of time.
- The GENERATE block starts creating transactions at time zero, but it can be configured to start at a later time using the C field. The D field can set a limit on how many transactions the block will create. Each transaction has a priority level and may carry parameters (data). The E field specifies the priority when the transaction is created.
- The parameters for each transaction can be different data types, such as signed integers (fullword, halfword, or byte) or signed floating-point numbers. If no specific type is assigned, the system defaults to creating transactions with halfword parameters. The C, D, and E fields are optional, depending on the simulation's needs.

GPSS: Succession of Events (1)

- The program keeps track of when each transaction is supposed to move through the system. It proceeds by completing all the actions scheduled for a specific time, making sure to perform tasks that can logically happen at that moment.
- If multiple transactions are ready to move, the program processes them based on their priority, and within the same priority, it follows a first-come, first-served approach.
- Once a transaction starts moving, the program continues to move it through the system until one of the following happens:
    1. The transaction enters an ADVANCE block with a non-zero action time. The program will focus on other transactions and return to this one once the action time is completed.

GPSS: Succession of Events (2)

2. The system's conditions prevent the transaction from proceeding. In this case, the transaction is "blocked" and stays at its current block. The program will detect when the blocking condition is resolved and will continue the transaction.
3. The transaction enters a TERMINATE block, which removes it from the simulation.
4. The transaction might be put on a chain for further processing.

- Once a transaction reaches a stopping point, the program checks if other transactions are ready to move at the same time. If all scheduled movements are complete, the program moves the clock forward to the time of the next event and repeats the process.

GPSS: Choice of Paths (1)

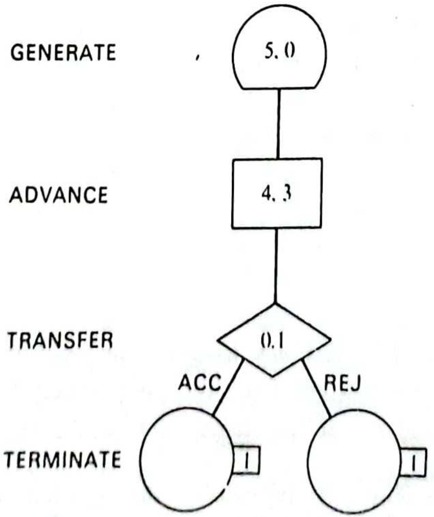
- The TRANSFER block allows a transaction to move to a block other than the next one in sequence. Typically, the decision is made between two blocks, referred to as "next blocks A and B." The method for making the choice is defined by the selection factor in Field A of the TRANSFER block. If no choice is needed, Field A is left blank, and the transaction will automatically move to next block A.
- To make a random choice, Field A can be set to a decimal fraction (S). The probability of moving to next block A will be 1-S, while the probability of moving to block B will be S. In conditional mode, Field A is set to "BOTH," allowing the transaction to select a path based on existing conditions. The transaction will move to next block A if that path is available, and to next block B if it is not. If neither path is available, the transaction will wait for one to become available, with preference given to block A if both become available at the same time.

GPSS: Simulation of a Manufacturing Shop (1)

A machine tool in a manufacturing shop is turning out parts at the rate of one every 5 minutes. As they are finished, the parts go to an inspector, who takes 4 ± 3 minutes to examine each one and rejects about 10% of the parts. Each part will be represented by one transaction, and the time unit selected for the problem will be 1 minute. Simulate it for 500 parts. Draw GPSS diagram and write code for the same.

```
GENERATE 5,0
ADVANCE 4,3
TRANSFER .1,ACC,REJ
ACC TERMINATE 1
REJ TERMINATE 1
START 500
```

GPSS: Simulation of a Manufacturing Shop (2)



GPSS: Simulation of a Manufacturing Shop (3)

Figure 3: Simulation of a Manufacturing Shop

A GENERATE block is used to simulate the machine's output, creating one transaction every five units of time. An ADVANCE block, with a mean of 4 and a modifier of 3, represents the inspection process. The inspection time can vary randomly between 1 and 7 minutes, with each value having an equal probability. After inspection, the transactions are directed to a TRANSFER block, which uses a selection factor of 0.1. This means that 90% of the parts will go to the ACC location (indicating accepted parts) and 10% will go to the REJ location (indicating rejected parts). Both the ACC and REJ locations are followed by TERMINATE blocks, which end the simulation for each part.

GPSS: Facilities and Storage (1)

In the simulated system, there are permanent entities, like equipment, that interact with the transactions. GPSS defines two types of permanent entities: facilities and storages.

- A facility is an entity that can only be used by one transaction at a time.
- A storage is an entity that can hold multiple transactions at once, up to a set limit.

A transaction using a facility can be interrupted or replaced by another transaction. Both facilities and storages can also become unavailable, such as when equipment breaks down, and they can be made available again once repairs are done.
There can be multiple instances of each entity, with a program-defined limit (usually up to 300). Each entity is given a unique identification number, and the number 0 is not allowed.

GPSS: Facilities and Storage (2)

GPSS uses several blocks to manage these entities:

- The SEIZE block allows a transaction to use a facility if it's available.
- The RELEASE block allows a transaction to stop using a facility.
- The ENTER block allows a transaction to occupy space in storage if it's available.
- The LEAVE block allows a transaction to release the space it was occupying.

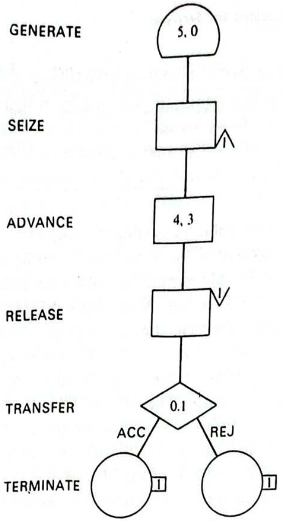
When using storages, the ENTER and LEAVE blocks can change the number of items in storage. If the B field is left empty, the storage count changes by 1. If a number is given, the storage count changes by that number.
Any number of blocks can be placed between the SEIZE and RELEASE blocks to simulate the actions of a transaction while using a facility, and similarly for using storage. To illustrate the use of these block types, con-

GPSS: Facilities and Storage (3)

sider 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 factthat only one part at a time can be inspected. A SEIZE block and a RELEASE block need to be added to simulate the engaging and disengaging of the inspector.

GPSS: Facilities and Storage (4)



GPSS: Facilities and Storage (5)

Figure 4: Simulation of a Manufacturing Shop; Facilities and Storage

```
GENERATE 5,0
SEIZE 1
ADVANCE 4,3
RELEASE
TRANSFER .1,ACC,REJ
ACC TERMINATE 1
REJ TERMINATE 1
START 500
```

GPSS: Facilities and Storage (6)

If there are multiple inspectors, they can be modeled as a group using a storage block with a capacity equal to the number of inspectors. The SEIZE and RELEASE blocks are replaced by ENTER and LEAVE blocks in this case. For example, if the inspection time is three times longer, three inspectors can be justified. A single inspector can be modeled by using a storage block with a capacity of 1 instead of a facility. The key difference between the two representations is that in a facility, only the transaction that seized it can release it, while in a storage, entry and exit can be performed independently by different transactions.

GPSS: Gathering Statistics (1)

- In GPSS, certain blocks are designed to gather statistics about system performance, including QUEUE, DEPART, MARK, and TABULATE blocks. These blocks help manage queues and tables, which are entities that track system activities.
- When transactions can't proceed because conditions aren't met, they wait in a queue. The QUEUE block increases the queue size, and the DEPART block decreases it. Transactions move in the queue based on priority, usually following a first-in, first-out rule.
- To measure how long transactions take to move through the system, the MARK and TABULATE blocks are used. MARK records the time when a transaction arrives, and TABULATE calculates how long it took to get to a certain block. This time is stored in a table.

GPSS: Gathering Statistics (2)

- In scenarios where all inspectors are busy, parts may accumulate, causing delays. A QUEUE block can be placed before the inspection block to store parts while they wait. When an inspector is available, the DEPART block removes parts from the queue and starts the inspection.
- The MARK and TABULATE blocks can also measure the time spent on inspection, excluding waiting time. If no MARK block is used, the time recorded is from when the transaction first entered the system.

In a more realistic scenario, if inspection takes too long, parts may accumulate on the inspector's workbench. To track this, it's important to measure the size of the work queue. This is done by placing a QUEUE block before the ENTER block and a DEPART block after the ENTER block to remove parts from the queue once inspection begins.

Transactions that do not need to wait for an inspector can proceed without delay. However, those that must wait will stay in the QUEUE block. The

GPSS: Gathering Statistics (3)

program automatically tracks how long these parts stay in the queue.

Additionally, MARK and TABULATE blocks are used to measure the time it takes to inspect the parts, excluding their waiting time. If a MARK block is not used, the TABULATE block will record the time since the transaction first entered the system.

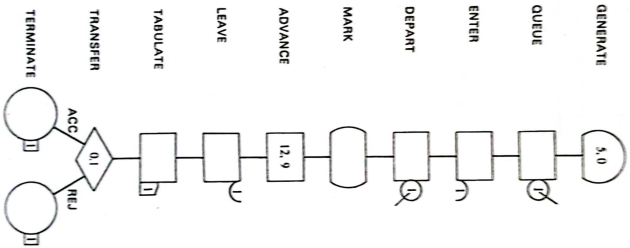


Figure 5: Gathering Statistics

GPSS: Gathering Statistics (4)

```
    GENERATE 5
    QUEUE 1
    ENTER 1
    DEPART
    MARK
    ADVANCE 4,3
    LEAVE
    TABULATE
    TRANSFER .1,ACC,REJ
    ACC TERMINATE 1
    REJ TERMINATE 1
    START 500
```

GPSS: Conditional Transfers (1)

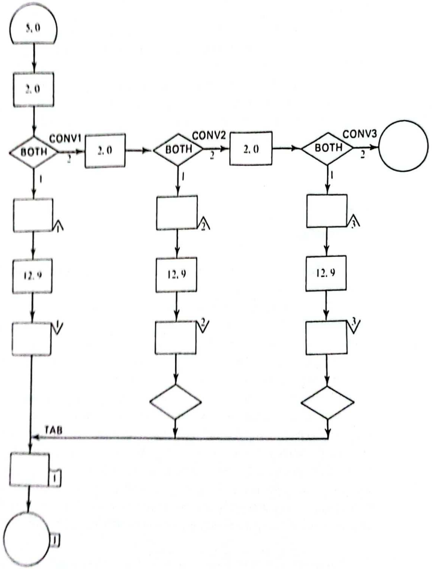
In this example, both conditional and unconditional transfer modes of the TRANSFER block are demonstrated. Imagine a system with three inspectors and a conveyor belt moving parts to them. It takes 2 minutes for a part to reach the first inspector. If the inspector is free, the part is inspected; if not, it moves to the second inspector, and then to the third, each 2 minutes apart. Parts that reach the third inspector or pass all inspectors are either inspected or lost. For simplicity, only the transit time is recorded, and parts' rejection by inspectors is not considered.

The conveyor movement is modeled by ADVANCE blocks, each with an action time of 2 minutes. As parts move along the conveyor, they enter a TRANSFER block to check if an inspector is available. The TRANSFER block uses a conditional selection factor (BOTH), leading to either a SEIZE block (if an inspector is free) or to the next stage (if the inspector is busy).

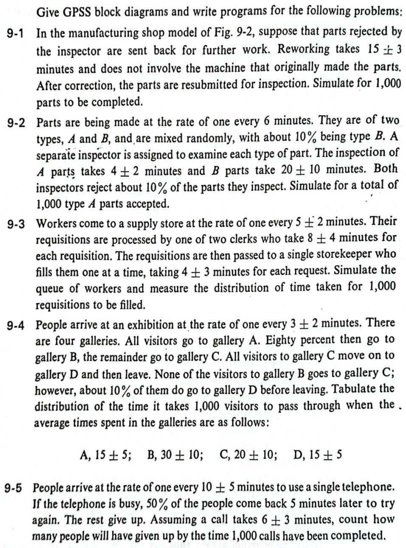
GPSS: Conditional Transfers (2)

Once inspection is complete, the parts enter a single TABULATE block to record the transit time. Since the GPSS system moves transactions in sequential order (from lower-numbered blocks to higher-numbered ones), only one of the three RELEASE blocks at the inspection phase can send transactions directly to the TABULATE block. The other RELEASE blocks will pass the transactions through TRANSFER blocks that unconditionally send them to the TABULATE block. Alternatively, three separate TABULATE blocks can be used at the exits of each RELEASE block, which would achieve the same results. This flexibility in using the same GPSS entity in multiple locations applies to all GPSS entities.

GPSS: Conditional Transfers (3)



Exercise



Give GPSS block diagrams and write programs for the following problems:

9-1 In the manufacturing shop model of Fig. 9-2, suppose that parts rejected by the inspector are sent back for further work. Reworking takes 15 ± 3 minutes and does not involve the machine that originally made the parts. After correction, the parts are resubmitted for inspection. Simulate for 1,000 parts to be completed.

9-2 Parts are being made at the rate of one every 6 minutes. They are of two types, *A* and *B*, and are mixed randomly, with about 10% being type *B*. A separate inspector is assigned to examine each type of part. The inspection of *A* parts takes 4 ± 2 minutes and *B* parts take 20 ± 10 minutes. Both inspectors reject about 10% of the parts they inspect. Simulate for a total of 1,000 type *A* parts accepted.

9-3 Workers come to a supply store at the rate of one every 5 ± 2 minutes. Their requisitions are processed by one of two clerks who take 8 ± 4 minutes for each requisition. The requisitions are then passed to a single storekeeper who fills them one at a time, taking 4 ± 3 minutes for each request. Simulate the queue of workers and measure the distribution of time taken for 1,000 requisitions to be filled.

9-4 People arrive at an exhibition at the rate of one every 3 ± 2 minutes. There are four galleries. All visitors go to gallery A. Eighty percent then go to gallery B, the remainder go to gallery C. All visitors to gallery C move on to gallery D and then leave. None of the visitors to gallery B goes to gallery C; however, about 10% of them do go to gallery D before leaving. Tabulate the distribution of the time it takes 1,000 visitors to pass through when the average times spent in the galleries are as follows:

A, 15 ± 5;   B, 30 ± 10;   C, 20 ± 10;   D, 15 ± 5

9-5 People arrive at the rate of one every 10 ± 5 minutes to use a single telephone. If the telephone is busy, 50% of the people come back 5 minutes later to try again. The rest give up. Assuming a call takes 6 ± 3 minutes, count how many people will have given up by the time 1,000 calls have been completed.

SIMSCRIPT - Introduction (1)

- `SIMSCRIPT` is a programming language specifically designed for *simulation modeling* and *simulation systems*. It is commonly used in areas such as *discrete-event simulation*, *manufacturing systems*, *computer networks*, and other complex system simulations.
- `SIMSCRIPT` allows users to model systems and processes, manage events, and analyze the resulting behaviors and performance metrics.
- SIMSCRIPT is a free-form, English-like simulation language created by Harry Markowitz and Bernard Hausner at the RAND Corporation in 1962.
- Originally implemented as a Fortran preprocessor on the IBM 7090, it was designed for large discrete-event simulations. SIMSCRIPT later influenced the development of Simula.

SIMSCRIPT - Introduction (2)

Key Features of SIMSCRIPT

- Discrete-Event Simulation: `SIMSCRIPT` excels at modeling systems where events occur at discrete points in time. It allows for the simulation of queues, resource allocation, and event handling in a dynamic environment.
- Object-Oriented Design: `SIMSCRIPT` incorporates object-oriented principles, making it possible to design systems using classes and objects. This approach helps in representing entities, their behaviors, and the interactions between them, much like how objects interact in the real world.
- Flexibility: `SIMSCRIPT` is versatile in modeling a wide variety of systems and processes, from simple to highly complex. The language allows for detailed control over the logic of the system being modeled.

SIMSCRIPT - Introduction (3)

- Event Scheduling: The core of `SIMSCRIPT` is its event-scheduling mechanism. Events are scheduled to occur at specific times, and the system updates its state in response to these events. This approach is essential for modeling dynamic systems like traffic flow or network congestion.
- Rich Built-In Libraries: `SIMSCRIPT` comes with libraries that help model common processes like queues, resource management, and time delays. This reduces the need for users to code basic functionalities from scratch.
- Simulation Time Control: `SIMSCRIPT` provides mechanisms to control the passage of simulation time and to track the progression of events, allowing users to simulate real-world systems step-by-step and observe how they evolve.

SIMSCRIPT - Introduction (4)

Structure of SIMSCRIPT Programs

`SIMSCRIPT` programs typically consist of the following components:

- Entities: These represent the components of the system, like customers, machines, or packets in a network.
- Events: Events trigger changes in the system and are typically scheduled at certain times.
- Resources: These are things the entities need to interact with, such as servers, machines, or bandwidth.
- Queues: Entities often wait in queues for resources to become available.
- State Variables: These track the conditions of entities or resources at any given time.

SIMSCRIPT - Introduction (5)

Applications of SIMSCRIPT

- Manufacturing Systems: Modeling production lines, resource allocation, machine breakdowns, and inventory management.
- Healthcare Systems: Simulating patient flows, staff allocation, and hospital resource utilization.
- Transportation Systems: Modeling traffic patterns, airport operations, and public transit systems.
- Telecommunications: Simulating network congestion, data packet routing, and service interruptions.

SIMSCRIPT System Concepts (1)

The viewpoint taken by the SIMSCRIPT user essentially corresponds to the general understanding of systems. In SIMSCRIPT, the system to be simulated is considered to consist of *entities* that have *attributes* which interact with *activities*. These interactions cause *events* that change the state of the system.

Entities and Attributes

Entities

Entities represent the components or objects that exist within the system. These can be things like customers, machines, packets in a network, etc. SIMSCRIPT have two types of entities: Permanent and Temporary.

- Permanent Entities: These entities exist throughout the simulation run. They do not get created and destroyed during the simulation execution. Examples could be system components that remain constant throughout the simulation period.

SIMSCRIPT System Concepts (2)

- Temporary Entities: These entities are created and destroyed during the simulation. They can represent entities like arriving customers or processes that only exist for a short time, from creation to destruction. Temporary entities are managed dynamically throughout the simulation.

Attributes

Attributes are properties associated with entities. These could include things like the age of a customer, the status of a machine, or the size of a packet. Attributes can be either permanent or temporary as well. Permanent attributes remain fixed for an entity's lifetime, while temporary attributes change dynamically based on the entity's actions or events in the simulation.

SIMSCRIPT System Concepts (3)

Sets in SIMSCRIPT

In SIMSCRIPT, sets are an important organizational structure used primarily with temporary entities.

- Temporary entities can be grouped into sets, making it easier to manage and interact with multiple entities at once. These sets allow the user to group entities based on common characteristics or roles, such as customers in a queue or machines under maintenance.
- Set Operations: The user can define, add, or remove entities from these sets. Facilities are provided for managing the entities in sets, and you can perform operations such as checking the size of a set, iterating over the entities, and moving entities into or out of a set as needed.

SIMSCRIPT System Concepts (4)

Activities

In SIMSCRIPT, activities are represented as processes that extend over time. These activities are marked by two main events: the beginning and (usually) the end.

- The beginning of an activity typically schedules the event that marks its end. For example, if a customer enters a queue, this event marks the beginning of the activity, and after some time, the event marking the end (such as being served) is scheduled.
- Each activity is associated with events that can influence other activities. For instance, the end of one activity may trigger the beginning of another activity.

SIMSCRIPT System Concepts (5)

Events

Events are actions that cause a change in the system's state. These events are triggered by specific conditions and cause transitions within the system.

- Events are typically described by event routines, which are subroutines programmed to handle specific types of events.
- Events can be classified as either or exogenous.

Types of Events

Endogenous (Internal) Events

- They originate from actions within the system itself. They are the result of internal system operations, such as the start or end of an activity.

SIMSCRIPT System Concepts (6)

- These events are usually triggered by scheduling statements within the event routines. For instance, the beginning of an activity might schedule an event marking the end of the activity.
- If the beginning or end of one activity causes another activity to start, the event routine that marks the end of the first activity will schedule the start of the second one.

Exogenous (External) Events

- They are external to the system and are triggered by actions or events from the environment surrounding the system.
- These events are defined by the user, who provides the necessary data, such as the time at which the event should occur.

SIMSCRIPT System Concepts (7)

- Unlike endogenous events, which are internal and often self-triggering, exogenous events require explicit input or data sets to function properly. These data sets can represent different types of external events, such as customer arrivals, machine breakdowns, or other system-related events that are governed by external factors.
- Exogenous events often require event routines to handle the specific actions that should occur when the event becomes due for execution.

Event Routines

Each type of event in SIMSCRIPT is associated with an event routine, which is a subroutine that defines the actions to be performed when the event occurs.

- Event routines are central to the operation of the simulation, and they must be programmed for each type of event (both endogenous and exogenous).

SIMSCRIPT System Concepts (8)

- Endogenous Event Routines: These are triggered by actions within the system, and are used for internal system transitions (e.g., starting or ending activities, scheduling new events based on internal conditions).
- Exogenous Event Routines: These event routines are triggered by external factors, such as data input by the user or external events like arrivals. They are particularly useful in scenarios where specific data needs to be read and processed at scheduled times (e.g., arrival times of customers based on a statistical distribution).

Bootstrap Method

In practice, external events, such as customer arrivals, are often generated using a statistical distribution, with the bootstrap method being one common approach.

SIMSCRIPT System Concepts (9)

- In the bootstrap method, given the statistical properties of inter-arrival times and other attributes, an endogenous event routine is used to continuously generate future arrivals. This process ensures that arrivals occur with the desired statistical distribution.
- An exogenous event routine is only necessary when specific data sets are required. For example, when historical data is involved or when multiple simulation runs are needed with identical data to reduce variance and ensure consistent results across different system designs.

SIMSCRIPT System Concepts (10)

Initialization and Exogenous Events

One part of SIMSCRIPT's automatic initialization is preparing the first exogenous event from each external data set. This initialization ensures that external events, such as the first arrival in a queue or the first breakdown of a machine, are scheduled at the correct time.

The routine initializes these events by reading the supplied external data, such as the arrival times of customers or other system-relevant events, and schedules them accordingly.

SIMSCRIPT System Concepts (11)

Summary of Concepts in SIMSCRIPT

- Entities:
  - Permanent Entities: Entities that persist throughout the simulation.
  - Temporary Entities: Entities that are created and destroyed dynamically during the simulation.
- Sets:
  - Temporary entities can be grouped into sets for better management and interaction.
  - Sets help organize entities based on their properties or roles.
- Event Routines:
  - Endogenous Events: Triggered by internal system actions and typically involve scheduling other events.
  - Exogenous Events: Triggered by external factors and require user-supplied data, typically involving the processing of historical data or specific event timings.

Table of Contents

SIMSCRIPT -Organization of A SIMSCRIPT Program (1)

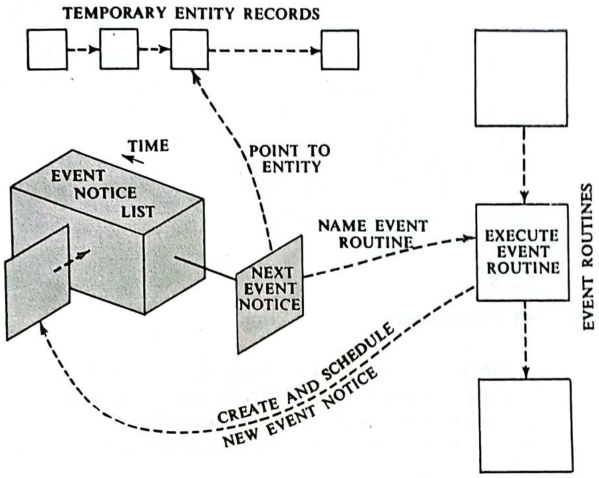


Figure 6: SIMSCRIPT Execution Cycle

SIMSCRIPT -Organization of A SIMSCRIPT Program (2)

1. Event Routines and Control Transfer
   - In the simulation, events are managed by routines, each of which corresponds to a specific event. These routines are ”closed,” meaning that control must be passed between them in a structured way.
   - Event notices are used to manage this control transfer. An event notice is created whenever an event is scheduled, containing key information such as the event time and the routine responsible for executing the event.
2. Chronological Ordering
   - Event notices are stored in chronological order to ensure that events are processed in the correct sequence based on their scheduled time.

SIMSCRIPT -Organization of A SIMSCRIPT Program (3)

- At each step in the simulation, the event routine processes all events that are due to occur at the current time. Once all such events have been executed, the simulation clock is updated to the next event's scheduled time, and control is passed to the next event routine.

3. Temporary Entities and Event Notices
   - Some events involve temporary entities (e.g., individual objects that may appear or disappear during the simulation). When such an entity is involved, the event notice will specify which particular instance of the entity is affected by the event.
   - In the case where the event involves a temporary entity, event notices help to manage the scheduling and processing of these dynamic objects.
4. Event Notice Lifecycle
   - Once an event notice has triggered the execution of a routine, it usually serves its purpose and is destroyed.

SIMSCRIPT -Organization of A SIMSCRIPT Program (4)

- However, there are cases where an event may generate new events. For instance, if an event involves a task that will result in a future event (like disconnecting a phone call), a new event notice must be created and scheduled for the future, continuing the simulation process.

5. Exogenous Events

- Exogenous events are external events that occur outside the scope of the simulation's internal routines (e.g., events triggered by outside factors). These events are handled in a similar manner to endogenous events, but they are managed through a separate process.
- Exogenous event statements specify the time of the event and the routine to be executed. These statements are sorted chronologically and processed when the event's scheduled time arrives.

SIMSCRIPT -Organization of A SIMSCRIPT Program (5)

Summary

- Event notices control the flow of the simulation by scheduling events and linking them to routines.
- Events are processed in chronological order, and once an event is processed, the simulation time is updated.
- Some events involve temporary entities, and their notices identify the specific entity.
- Event notices are usually destroyed after execution but may be rescheduled if the event generates another future event.
- Exogenous events follow a similar structure to endogenous events but are managed separately.

Names and Labels in SIMSCRIPT (1)

In SIMSCRIPT, naming conventions for entities and attributes are flexible, allowing users to create descriptive names for various elements of the simulation. Here's a detailed breakdown of the naming rules:

1. Entity Names
   - Format: Can consist of letters and digits, as long as there is at least one letter.
   - Example: An entity representing people could be named `PERSON`.
   - Compound Names: It's possible to use periods (`.`) to create compound names, which help organize and define hierarchical relationships.
   - Example: `PERSON.AGE` for the age attribute of a person.
2. Attribute Names
   - Format: Like entity names, attributes can be named with a combination of letters and digits, with at least one letter.

Names and Labels in SIMSCRIPT (2)

- Example: If a person's age and education are attributes, they could be named `AGE` and `EDUCATION`, or more specifically as `PERSON.AGE` and `PERSON.EDUCATION` to specify they belong to the `PERSON` entity.

3. Label Names (for programming statements)
   - Format: Labels can be any combination of letters and numbers without the restriction of requiring at least one letter, unlike entity or attribute names.
   - Compound Names: Labels can also use periods to form compound labels.
   - Enclosed in Quotes: Labels are identified by being enclosed in single quotation marks (`'`).
   - Example: `'START_SIMULATION'` could be a label for the start of the simulation process.
4. Character Limitations

Names and Labels in SIMSCRIPT (3)

- Practical Limit: While names can technically be of any length for documentation purposes, most machine implementations recognize only a limited number of initial characters (typically eight). This is something to keep in mind when naming entities and attributes for compatibility with the system.

Examples

- Entities: `PERSON`, `EMPLOYEE`, `PRODUCT`
- Attributes: `PERSON.AGE`, `PRODUCT.COST`, `EMPLOYEE.SALARY`
- Labels: `'BEGIN_SIMULATION'`, `'END_PROCESS'`

Table of Contents

SIMSCRIPT Programs (1)