CONTENTS

SYLLABUS

C-136 MODELING AND SIMULATION

Unit-I

System definition and components, stochastic activities, continuous and discrete Systems, System modeling, types of models, static and dynamic physical models, Statics and dynamic mathematical models, Full corporate model, types of system study.

Unit-II

System simulation, Why to simulate and when to simulate, Basic nature of simulation, technique of simulation, comparison of simulation and analytical methods, types of system simulation, real time simulation, hybrid simulation, simulation of pure-pursuit problem single-server queuing system and an inventory problem, Monte Carlo simulation, Distributed Lag models, Cobweb model.

Unit-III

Simulation of continuous systems, analog vs. digital simulation, simulation of water reservoir system, simulation of a servo system, simulation of an autopilot, Discrete system Simulation, Fixed time-step vs. event-to-event model, generation of random numbers, Test for randomness, Generalization of non-uniformly distributed random numbers, Monte-Carlo computation vs. stochastic simulation.

Unit-IV

System dynamics, exponential growth models, exponential decay models, modified exponential growth models, logistic curves, generalization of growth models, System dynamics diagrams, Feedback in Socio-Economic systems, world model.

Unit-V

Simulation of PERT networks, Critical path computation, uncertainties in Activity duration, Resource allocation and consideration. Simulation software, Simulation languages, continuous and discrete simulation languages, Expression based languages, object-oriented simulation, general-purpose vs. application-oriented simulation packages, CSMP-III, MODSIM-III.

NOTES

UNIT I SYSTEMS CONCEPTS AND MODELING

★ STRUCTURE ★

1.0 LEARNING OBJECTIVES

After going through this unit, you will be able to:

- illustrate modeling and simulation concepts as well as system concept.
- explain about principle used in modeling.
- enumerate model verification and its validation.
- define types of models.

1.1 INTRODUCTION

Computer system users, administrators, and designers usually have a goal of highest performance at lowest cost. Modeling and simulation of system design trade off is good preparation for design and engineering decisions in real world jobs. In this subject we study modeling and simulation of a variety of systems.

Simulation is one of the most powerful tools available to decision-makers responsible for the design and operation of complex processes

NOTES

and systems. It makes possible the study, analysis and evaluation of situations that would not be otherwise possible. In an increasingly competitive world, simulation has become an indispensable problem solving methodology for engineers, designers and managers. The simulation discipline has now expanded to include modeling of systems that are human-centered (like commercial, economical, and social) thus containing a large amount of uncertainty. Those new fields of applications make modeling and simulation a dynamically expanding discipline. However, there is a growing gap between the new problems and the methodology. In particular, more robust research is required in the area of continuous simulation methodology and numerical methods. Another important question is model validation: it is difficult to prove that the model used is absolutely valid. Through examining the system dynamics methodology, which is over fifty years old, and it is still used in many application fields. However, less attention is given to the model validity. Modeling and simulation is somewhere between science and art, frequently resembling the art of misapprehension.

Computer system users, administrators, and designers usually have a goal of highest performance at lowest cost. Modeling and simulation of system design trade off is good preparation for design and engineering decisions in real world jobs. In this subject we study modeling and simulation of a variety of systems.

Modeling and Simulation is a discipline, it is also very much a form of art. One can learn about riding a bicycle from reading a book. To really learn to ride a bicycle one must become actively engaged with a bicycle. Modeling and Simulation follows much the same reality. One can learn much about modeling and simulation from reading books and talking with other people. Skill and talent in developing models and performing simulations is only developed through the building of models and simulating them. From the interaction of the developer and the models emerges an understanding of what makes sense and what does not.

The terms model and system are key components of simulation. By a model we mean a representation of a group of objects or ideas in some form other than that of the entity itself. By a system we mean a group or collection of interrelated elements that cooperate to accomplish some stated objective. One of the real strengths of simulation is the fact that we can simulate systems that already exist as well as those that are capable of being brought into existence, *i.e.,* those in the preliminary or planning stage of development. Dynamic modeling in organizations is the collective ability to understand the implications of change over time. This skill lies at the heart of successful strategic decision process. The availability of effective visual modeling and simulation enables the analysts and the decision-makers to boost their dynamic decision by rehearsing strategy to avoid hidden pitfalls.

System Simulation is the mimicking of the operation of a real system,

NOTES

such as the day-to-day operation of a bank, or the running of an assembly line in a factory, or the value of a stock portfolio over a time period, or the staff assignment of a hospital or a security company, in a computer. Instead of building extensive mathematical models by experts, the readily available simulation software has made it possible to model and analyze the operation of a real system by non-experts, who are managers but not programmers.

A simulation is nothing but the execution of a model, represented by a computer program that gives information about the system being investigated. The simulation approach of analyzing a model is opposed to the analytical approach, where the method of analyzing the system is purely theoretical. As this approach is more reliable, the simulation approach gives more flexibility and convenience. The activities of the model consist of events, which are activated at certain points in time and in this way affect the overall state of the system. The points in time that an event is activated are randomized, so no input from outside the system is required. Events exist autonomously and they are discrete so between the executions of two events nothing happens. The SIMSCRIPT (a simulation language) provides a process-based approach of writing a simulation program. With this approach, the components of the program consist of entities, which combine several related events into one process.

In the field of simulation, the concept of principle of computational equivalence has beneficial implications for the decision-maker. Simulated experimentation accelerates and replaces effectively the wait and watch anxieties in discovering new insight and explanations of future behavior of the real system.

With the integration of artificial intelligence, agents and other modeling techniques, simulation has become an effective and appropriate decision support for the managers. For example, in a consumer retail environment it can be used to find out how the roles of consumers and employees can be simulated to achieve peak performance. It is apparent that there are many problems of real life that cannot be represented mathematically due to the stochastic nature of the problem, the conflicting ideas needed to properly describe the problem under study, or the complexity in problem formulation. Therefore under such circumstances simulation is the most often used tool. The analysts and designers of physical systems have long applied the simulation techniques and these now have become important tools for dealing with the complex problems in real life.

Most but not all digital integrated circuits manufactured today are first extensively simulated before they are manufactured to identify and correct design errors. Simulation early in the design cycle is important because the cost to repair mistakes increases dramatically the later in the product life cycle that the error is detected. Another

NOTES

important application of simulation is in developing virtual environments, for example in training. Simulations generate dynamic environments with which users can interact as if they were really there. Such simulations are used extensively today to train military personnel for battlefield situations, at a fraction of the cost of running exercises involving real tanks, aircraft, etc.

Von Neuman and Stanislaw Ulàm probably developed first important application of simulation for determining the complicated behavior of neutrons in a nuclear shielding problem being to complex for mathematical analysis. Computer simulation provides a means to take off the behaviors of complex real systems both quickly and economically. Simulation models can expeditiously compare the outcomes of alternatives before selecting a course of action. Simulation models can also provide a dynamic virtual environment for training. All simulation models require the mathematical representation of a real system that exists, or could exist, in time and space. A computational representation of that system then links inputs to outputs through the system architecture.

Computer-based modeling and simulation is used extensively for development of many complex, large-scale systems such as networks, information systems, and physical systems. Modeling concepts, theories, and methods provide a foundation for characterizing structure and behavior of dynamical systems at varying levels of details. These models can be constructed and subsequently simulated. Since dynamical systems can be described using alternative modeling and simulation approaches, it is important to understand their strengths and appropriateness.

The objective of this subject is to learn about heavy construction equipment, methods, modeling, and simulation. You will learn how to choose and configure equipment for different purposes such pure pursuit problem, reservoir, inventory control, queueing system, etc.

It is often said that computers are revolutionizing science and engineering. By using computers we are able to construct complex engineering designs such as space shuttles. We are able to compute the properties of the universe, as it was fractions of a second after the big bang. Our ambitions are ever increasing. We want to create even more complex designs such as better spaceships, cars, medicines, computerized cellular phone systems, etc. We want to understand deeper aspects of nature. These are just a few examples of computer-supported modeling and simulation.

This text presents an object-oriented component-based approach to computer-supported mathematical modeling and simulation through the powerful Modelica language and its associated technology. Modelica can be viewed as an almost-universal approach to high-level computational modeling and simulation, by being able to represent a range of application areas and providing general notation as well as powerful abstractions and efficient implementations.

1.2 MODELING AND SIMULATION PROJECTS

Several projects have been developed so far and several are under progress. The main objectives of the modeling and simulation project being developed are to:

- Learn about the potential of modeling and simulation tools in construction
- Learn how to use simple modeling and simulation software
- Analyze a specific site operation in order to gain insight into its design

The modeling and simulation project includes the following tasks:

1. Use of modeling and simulation software package to accurately model and simulate an operation from one site to which a field trip was organized. This will require learning about basic modeling approaches, collecting site data, and learning how to use the software. There are usually classic models available. Accuracy of the model must be verified.
2. Perform a sensitivity analysis of the operation.
3. Suggest possible improvements and/or alternatives to the operation based on analysis.
4. Evaluate the software used based on its ease to learn and usefulness for different applications.
5. Documentation of the model, simulation of results, sensitivity analysis, and final conclusions and recommendations in a project report.

1.3 THE SYSTEM CONCEPT

Before simulation we need a system. Let us define a system first. What is a system? A system can be almost anything. A system can contain subsystems, and the subsystem may again contain subsystems. A possible definition of systems might be: System is a well defined object in the Real World under specific conditions, only considering specific aspects of its structure and behaviour, or a system is an object or collection of objects whose properties we want to study. A system can also be defined as the collection of objects that act and interact together toward some logical end. Our wish to study selected properties of objects is central in this definition. The selection and definition of what constitutes a system is somewhat arbitrary and must be guided by what the system is to be used for.

NOTES

A system can be understood as an entity, which maintains its existence through the interaction of its parts. Model is a simplified representation of the actual system intended to promote understanding. Whether a model is a good model or not depends on the extent to which it promotes understanding. Since all models are simplifications of reality there is always a trade-off as to what level of detail is included in the model. If too little detail is included in the model one runs the risk of missing relevant interactions and the resultant model does not promote understanding. If too much detail is included in the model the model may become overly complicated and actually exclude the possibility of the development of understanding.

A system is a potential source of data. It can be defined as a list of variables. There are variables that are generated by the environment, and which influence the behavior of the system. These variables are called the *input* of the system. There are some other variables that are determined by the system, and that in turn influence the behaviour of its environment. These variables are called *outputs* of the system.

Some other definition of systems can also be adapted. While defining the system, there are certain tasks that must be done. Among these are:

- Divide the system into logical subsystems.
- Define the entities, which will flow through the system.
- For each subsystem, define the stations (locations where something is done to or for the entities).
- Define the basic flow patterns of entities through the stations using flow diagrams.
- Define alternative designs for the system, which are to be considered.
- Develop flow charts to show the routing logic for flexible paths.

System Terminology

We live in a world of systems driven by causes and affects. Those systems include inventory, production, financial, chemical, biological, thermodynamic or workflow. Systems can be modeled as nodes representing system variables and connecting lines representing causal effects. The changing value of one variable can cause another to increase or decrease. Understanding how a system really works is the first step towards using, improving, automating or explaining it to others.

Causal Loop Diagrams (CLDs) are used to model the dynamic systems. It is the simple diagram notation of nodes and lines identifies the important variables in a system and how they interact. Facts about the system are used to parameterize a causal loop diagram. Each node becomes a variable that identifies a quantifiable property of the system that changes over time. Each line can have equations or rules that formalize how one variable affects another.

The parameterized models must have all the information needed to simulate the dynamic response of the system over a series of time increments. The model is declarative in that the diagram, variables and equations just declare facts about the system. Let us define various factors of a system:

- **State:** Collection of variables and their values necessary to characterize a system at a particular time. It can also be defined as a variable characterizing an attribute in the system such as level of stock in inventory or number of jobs waiting for processing.
- **Event:** A change in system state, or an occurrence at a point in time which may change the state of the system, such as arrival of a customer or start of work on a job.
- **Entity:** An object that passes through the system.
- **Queue:** A queue is not only a physical queue of people, it can also be a task list, a buffer of finished goods waiting for transportation or any place where entities are waiting for something to happen for any reason.
- **Creating:** Creating is causing an arrival of a new entity to the system at some point in time.
- **Scheduling:** Scheduling is the act of assigning a new future event to an existing entity.
- **Random variable:** A random variable is a quantity that is uncertain, such as inter-arrival time between two incoming flights or number of defective parts in a shipment.
- **Random variate:** A random variate is an artificially generated random variable.
- **Distribution:** A distribution is the mathematical law, which governs the probabilistic features of a random variable.

Entity States

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

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

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

NOTES

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

The Condition-delayed State. The Condition-delayed State is the state of entities delayed until some specified condition comes about, *e.g.,* a *part entity* might wait in the condition-delayed state until its turn comes to use a machine. Condition-delayed entities are removed *automatically* from the Condition-delayed state when conditions permit.

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

Entity Management Structures

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

The Active Entity. The active entity forms an unnamed *list* consisting only of the active entity. The Active-State entity moves nonstop until encountering an operation that puts it into another state (transfers it to another list) or removes it from the model. A Ready-State entity then becomes the next Active-State entity. Eventually there is no possibility of further action at the current time. The EMP then ends and a Clock Update Phase begin.

The Current Events List. Entities in the Ready State are kept in a single list here called the *current events list* (CEL). Entities migrate to the current events list from the future events list, from delay lists, and from user-managed lists. In addition, entities cloned from the Active-State entity usually start their existence on the current events list.

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

NOTES

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

The preceding statement assumes there are not other entities on the FEL whose move time matches the clock's updated value. In the case of move-time ties, some tools will transfer all the time-tied entities from the FEL to the CEL during a single CUP, whereas other tools take a "one entity transfer per CUP" approach. Languages that work with internal entities usually use the FEL to support the timing requirements of these entities. The FEL is typically composed both of external and internal entities in such languages.

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

If a delay can be related easily to events in the model that might resolve the condition, then related waiting can be used to manage the delay list. For example, suppose a machine's status changes from busy to idle. In response, the software can automatically remove the next machine-using entity from the appropriate delay list and put it in the Ready State on the current events list. Related waiting is the prevalent approach used to manage conditional delays. If the delay condition is too complex to be related easily to events that might resolve it, polled waiting can be used. With polled waiting the software checks routinely to see if entities can be transferred from one or more delay lists to the Ready State. Complex delay conditions for which polled waiting can be useful include Boolean combinations of state changes, *e.g.*, a part supply runs low or an output bin needs to be emptied.

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

A Simple Example

Let us consider building a simulation fuel station with a single pump served by a single service man. Assume that arrival of vehicles as well their service times are random. At first identify the:

NOTES

- **States:** number of vehicles waiting for service and number of vehicles served at any moment
- **Events:** arrival of vehicles, start of service, and end of service
- **Entities:** these are the vehicles in this example
- **Queue:** the queue of vehicles in front of the pump, waiting for service
- **Random Realizations:** inter-arrival times, service times, etc.
- **Distributions:** we shall assume exponential distributions for both the inter-arrival time and service time.

Next, specify what to do at each event. The above example would look like this, at event of entity arrival, Create next arrival. If the server is free, send entity for start of service. Otherwise it joins the queue. At event of service start, Server becomes occupied. At event of service end, Server becomes free. If any entities waiting in queue then: (i) remove first entity from the queue; and (ii) send it for start of service.

Some initiation is still required, for example, the creation of the first arrival. Lastly, the above is translated into code. This is easy with an appropriate library that has subroutines for creation, scheduling, proper timing of events, queue manipulations, random variate generation, and statistics collection.

How to simulate? Besides the above, the simulation program records the number of vehicles in the system before and after every change, together with the length of each event.

Systems and Experiments

Experimentation is the physical act of carrying out an experiment. An experiment may interfere with system operation (influence its input and parameters) or it may not. As such, the experimentation environment may be seen as a system in its own right (which may in turn be modeled by a lumped model). Also, experimentation involves observation. Observation yields measurements. A system is given, what reasons can there be to study a system? There are many answers to this question but we can distinguish two major motivations:

- Study a system to understand it in order to build it. This is the engineering point of view.
- Satisfy human interest, *e.g.,* to understand more about nature.
- **System:** That which is to be described, analyzed and controlled, anything of interest, which is to be described in detail. Often defined as a collection of objects enclosed by a boundary, but this is not essential and the boundary may be conceptual rather than tangible.

- **Environment:** *All that is external to the system. Everything* else of interest, but which will not be described in detail. Commonly conceived, as external to the system, but again, this is not essential.
- **Open and Closed Systems:** The behavior of an open system may depend upon its environment; *i.e.,* the two interact. A closed system does not interact with its environment.
- **System Variable:** A quantity, used to describe the system, which may change with time or space.
- **System Input:** A quantity that is prescribed or imposed on the system by the environment; *i.e.,* an independent variable.
- **System Output:** *Any* system variable of interest is system's output.
- **State Determined Systems (SDS):** A class of systems fully determined by a finite set of *state variables.*
- **State:** A minimal, complete and independent set of state variables that uniquely describe the system.
- **State Equations:** To describe a state-determined system's behavior uniquely for all $t > t_0$ it is sufficient to have:

  (i) Value of a finite set of variables $(x_1, x_2, x_3, \ldots, x_n)$ at t_0,

  (ii) Value of a finite set system inputs $(u_1, u_2, u_3, \ldots, u_n)$ for all $t > t_0$, and

  (iii) A set of state equations: $y_m = g_m(x_1, x_2, \ldots, x_n, u_1, u_2, u_3, \ldots u_r, t)$

$$dx_1/dt = f_1(x_1, x_2, \ldots, x_n, u_1, u_2, u_3, \ldots u_r, t)$$
$$dx_2/dt = f_2(x_1, x_2, \ldots, x_n, u_1, u_2, u_3, \ldots u_r, t)$$
$$dx_3/dt = f_3(x_1, x_2, \ldots, x_n, u_1, u_2, u_3, \ldots u_r, t)$$
$$\vdots$$
$$dx_n/dt = f_n(x_1, x_2, \ldots, x_n, u_1, u_2, u_3, \ldots u_r, t)$$

- **Output equations:** Any output variables of a state-determined system may be expressed as functions of its state and input variables:

$$y_1 = g_1(x_1, x_2, \ldots, x_n, u_1, u_2, u_3, \ldots u_r, t)$$
$$y_2 = g_2(x_1, x_2, \ldots, x_n, u_1, u_2, u_3, \ldots u_r, t)$$
$$y_3 = g_3(x_1, x_2, \ldots, x_n, u_1, u_2, u_3, \ldots u_r, t)$$
$$\vdots$$
$$y_m = g_m(x_1, x_2, \ldots, x_n, u_1, u_2, u_3, \ldots u_r, t)$$

NOTES

Vector Notation

A more compact notation in terms of vectors is given as follows:

- **State Space:** An abstract *n*-dimensional space defined by the state variables.
- **State Vector:** A point in state space defined by a complete set of state variables.
- **Input Space:** An abstract r-dimensional space defined by the input variables.
- **Input Vector:** A point in input space defined by a complete set of input variables.
- **Output Space:** An abstract m-dimensional space defined by the output variables.
- **Output Vector:** A point in output space defined by a complete set of output variables.

Natural and Artificial Systems

As, a system, can be described as orderly interconnections of parts into a meaningful whole. As we are concerned with the classifications of systems, we can easily recognize three broad categories of systems:

Natural Systems—The systems whose operation is beyond the man's control are called natural systems, like solar system.

Man Made Systems—The systems, which are devised and operated by human, like aircraft flight control system.

Hybrid System—The systems, which are only partially controllable by human being, like artificial rain production system.

A system can occur naturally, *e.g.,* the universe, and it can also be artificial such as a space shuttle, or a mix of both (hybrid). For example, the house in with solar-heated valve warm water is an artificial system, *i.e.,* manufactured by humans. If we also include the sun and clouds in the system it becomes a combination of natural and artificial components.

At this point, we must be clear about how a system is to be defined for specific simulation. Our first impulse is to point at the pendulum and to say "the system is that thing there". This method, however, has a fundamental disadvantage: every material object contains no less than infinity of variables, and therefore, of possible systems. The real pendulum, for instance, has not only length and position; it has also mass, temperature, electric conductivity, chemical impurities, crystalline structure, some radioactivity, velocity, reflecting power, tensile strength, a surface film of moisture, bacterial contamination, an optical absorption, elasticity, shape, specific gravity, etc. Any suggestion that we should study all the facts is unrealistic, and actually the

attempt is never made. The necessary point is that we should pick out and study the facts that are relevant to some main interest that is already given.

If the system is completely artificial, we must be highly selective in its definition depending on what aspects we want to study for the moment. An important property of systems is that they should be *observable*. Some systems, but not large natural systems like the universe, are also *controllable* in the sense that we can influence their behavior through inputs:

- The *inputs* of a system are variables of the environment that influence the behavior of the system. These inputs may or may not be controllable by us.
- The *outputs* of a system are variables that are determined by the system and may influence the surrounding environment.

In many systems the same variables act as both inputs and outputs. We talk about a *causal* behavior if the relationships or influences between variables do not have a causal direction, which is the case for relationships described by equations. For example, in a mechanical system the forces from the environment influence the displacement of an object, but on the other hand the displacement of the object influences the forces between the object and environment. What is input and what output is in this case, is primarily a choice by the observer, guided by what is interesting to study, rather than a property of the system itself.

Experiment with System

An experiment can be defined as a process of extracting data from system by exerting it through its inputs. Experimenting with a system thus means to make use of its property of being controllable and observable. To perform an experiment on the system means to apply a set of external conditions to the accessible inputs, and to observe the reaction of the system to these inputs by considering the route behavior of the accessible outputs.

Observation is essential in order to study a system according to our definition of system. We must atleast be able to observe some outputs of a system. We can learn even more if it is possible to exercise a system by controlling its inputs. This process is called *experimentation*, like the definition below says:

An *experiment* is the process of extracting information from a system by exercising its inputs. To perform an experiment on a system it must be both controllable and observable. We apply a set of external conditions to the accessible inputs and observe the reaction of the system by measuring the accessible outputs.

NOTES

One of the disadvantages of the experimental method is that for a large number of systems many inputs are not accessible and controllable. These systems are under the influence of inaccessible inputs, sometimes called *disturbance inputs*. Likewise, it is often the case that many really useful possible outputs are not accessible for measurements; these are sometimes called *internal states* of the system. There are also a number of practical problems associated with performing an experiment, like:

- *The experiment might be too expensive*: investigating ship durability by building ships and letting them collide is a very expensive method of gaining information.
- *The experiment might be too dangerous*: training nuclear plant operators in handling dangerous situations by letting the nuclear reactor enter hazardous states is not advisable.
- *The system needed for the experiment might not yet exist*: This is typical of systems to be designed or manufactured.

The shortcomings of the experimental method lead us over to the model concept. If we make a model of a system, this model can be investigated and may answer many questions regarding the real system if the model is realistic enough. One of the major disadvantages of experimenting with real or actual system is that the systems are under the influence of a large number of additional inaccessible inputs and a number of real useful outputs, which are not accessible through measurements either.

Preliminary Data Preparation and Experimental Design

We can define simulation as being experimentation via a model to gain information about a real world process or system. It then can follow that we must concern ourselves with the strategic planning of how to design experiments that will yield the desired information at the lowest cost. The design of experiments comes into play at two different stages of a simulation study. It first comes into play very early in the study, before the first line of code has been written and before the finalization of the design of the model. As early as possible, we want to select the measures of effectiveness to be used in the study, what factors we are going to vary, how many levels of each of those factors we will investigate and the number of samples we will need in order to carry out the entire experiment. We can calculate ahead of time the sample sizes needed. If the number is large then we know that the model must run very fast and let this influence the design of the model. Having this fairly detailed idea of the experimental plan early, allows the model to be better planned to provide efficient generation of the desired data. We need data to drive our simulation.

model. Every simulation study involves input data gathering and analysis. Stochastic systems contain numerous sources of randomness. The analyst must therefore be concerned about what data to use for input to the model for things such as the inter-arrival rate of entities to the system, processing times required at various stations, time between breakdowns of equipment, rejection rates, travel times between stations etc. Having good data to drive a model is just as important as having sound model logic and structure. Data gathering is usually interpreted to mean gathering numbers, but gathering numbers is only one aspect of the problem. The analyst must also decide what data is needed, what data are available and whether it is pertinent, whether existing data are valid for the required purpose, and how to gather the data.

In real world simulation studies, the gathering and evaluation of input data is very time consuming and difficult. Up to one third of the total time used in the study is often consumed by this task. Depending upon the situation, there are several potential sources of data. These include:

- Historical records
- Observational data
- Similar systems
- Operator estimates
- Vendor's claims
- Designer estimates
- Theoretical considerations.

Each of these sources has potential problems. Even when we have copious data, it may not be relevant. For example we may have sales data when we need demand data. In other cases we may have only summary statistics. When historical data does not exist, the problem is even more difficult. In such cases we must estimate both the probability distribution and the parameters based upon theoretical considerations.

1.4 CONCEPT OF MODEL

The essence of the art of modeling is abstraction and simplification. We want to design a model of the real system that neither oversimplifies the system to the point where the model becomes trivial (or worse misleading) nor carries so much detail that it becomes clumsy and prohibitively expensive to build and run. The tendency among inexperienced modelers is to try to include too much detail. One should always design the model around the questions to be answered rather than try to imitate the real system exactly. In every group or collection of entities, there exists a vital few and a trivial many. In fact 80% of the

NOTES

behavior can be explained by the action of 20% of the components. Our problem in designing the simulation model is to make sure that we correctly identify the vital few components and include them in our model.

Modeling is used to represent/re-use/exchange knowledge about system structure and behavior. A model is a simplified representation of a system at some particular point in time or space intended to promote understanding of the real system. A model (M) for a system (S) and an experiment (E) is anything to which experiment (E) can be applied in order to answer questions about system (S).

Above definition does not imply that a model is a computer program. It can also be a piece of hardware or simply an understanding of how a particular system works. Given the previous definitions of system and experiment, we can now attempt to define the notion of model:

A *model* of a system is anything an experiment can be applied to in order to answer questions about that *system*. This implies that a model can be used to answer questions about a system without doing experiments on the real system. Instead we perform a kind of simplified experiments on the model, which in turn can be regarded as a kind of simplified system that reflects properties of the real system. In the simplest case a model can just be a piece of information that is used to answer questions about the system.

Given this definition, any model also qualifies as a system. Models, just like systems, are hierarchical in nature. We can cut out a piece of a model, which becomes a new model that is valid for a subset of the experiments for which the original model is valid. A model is always related to the system it models and the experiments it can be subject to. A statement such as "a model of a system is invalid" is meaningless without mentioning the associated system and the experiment. A model of a system might be valid for one experiment on the model and invalid for another. The term model *validation* always refers to an experiment or a class of experiment to be performed.

We talk about different kinds of models depending on how the model is represented:

- **Physical Model:** This is a physical object that mimics some properties of a real system, to help us answer questions about that system. For example, during design of artifacts such as buildings, reservoirs, airplanes, etc., it is common to construct small physical models with same shape and appearance as the real objects to be studied, *e.g.,* with respect to their smooth properties and aesthetics.
- **Mathematical Model:** A description of a system where the relationships between variables of the system are expressed in

mathematical form. Variables can be measurable quantities such as size, length, weight, volume, temperature, unemployment level, bit rate, information flow, etc. Most laws of nature are mathematical models in this sense. For example, Ohm's law describes the relationship between current and voltage for a resistor; Newton's laws describe relationships between velocity, acceleration, mass, force, etc.

- **Mental Model:** A statement like "a person is reliable" helps us answer questions about that person's behavior in various situations, such kinds of models are mental models.
- **Verbal Model:** This kind of model is expressed in words. For example, the sentence "More accidents will occur if the speed limit is increased" is an example of a verbal model. Expert systems is a technology for formalizing verbal models.

The kinds of models that we primarily deal with in this text are mathematical models represented in various ways, *e.g.*, as equations, functions, simulation programs, etc. Artifacts represented by mathematical models in a computer are often called virtual prototypes. The process of constructing and investigating such models is virtual prototyping. Sometimes the term physical modeling is used also for the process of building mathematical models of physical systems in the computer if the structuring and synthesis process is the same as when building real physical models.

Why Models

The use of models is an essential part of the decision making process. These models range from the mental models buried in the mind of the decision maker and not necessarily visible, to the explicit large scale models used to explore the consequences of specific decisions or phenomena affecting outcomes of a given model. To some extent perceived utility of the model will be influenced by the complexity of the model and it then follows that the drive for increased utility and complexity in the model will lead to the situation where the person making the decision is unable to understand the processes followed by the model in producing its outcomes. The need to establish validity and the process of model verification have been debated in the simulation area for many years, and with the development of animated interfaces it is now possible to adopt more comprehensive processes of validation and verification to be followed leading to higher levels of credibility for the model. Simulation therefore could occupy a more prominent position in the tool-kit of decision makers as the animation interface and ease of model development continues to advance over the next ten years. Development of simpler and more powerful interfaces,

NOTES

often referred to as simulators, has led to the situation where simulation is no longer the technique of last resort but is a technique which is available to engineers, designers and managers.

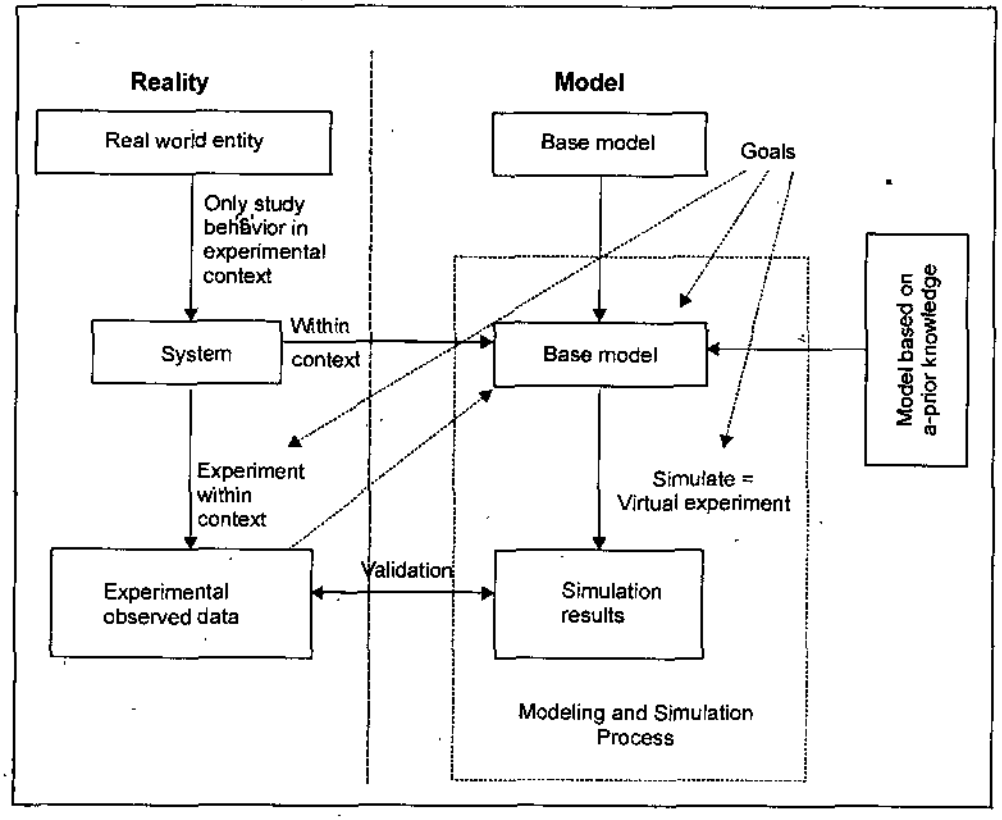

Fig. 1.1 Relation in reality and model

In this section we would like to discuss simulation as the process of designing a computer based model of a reference system, which may be real or proposed, and conducting experiments with this model for the purpose of understanding the behavior of the reference system and/or evaluating various strategies for the design or operation of the reference system. The specific area of simulation which is the focus of this study is discrete-event simulation (DES) which is the modeling of systems in which the state of variable changes only at a discrete set of points of time. The relation in reality and the developed model is explained in Fig. 1.1.

1.5 PRINCIPLES USED IN MODELING

Methods. Methods Lack of adequate modeling methods is one of the most serious shortfalls in using Modeling and Simulation. In order to maximize the potential of Modeling and Simulation technologies for commercial manufacturing and defense acquisition, basic research must be undertaken to improve understanding of modeling methods

and characteristics, including scalability, multi-resolution modeling, semantic consistency, modeling complexity, fundamental limits of modeling and computation, and uncertainty. Scalability is the attribute of a system's architecture that pertains to the behavior and performance of the system as the size, complexity, and interdependence of its elements or applications increase. Difficulties in dealing with large-scale software systems are well documented. Techniques that work for small systems often fail markedly when the scale is increased significantly. To be upwardly scalable, a system must assure consistency in both the functionality and the quality of the services it provides as the number of its users increases indefinitely. To scale by a million, an application's storage and processing capacity would have to be able to grow by a factor of 1 million just by adding more resources.

Components. Traditional modeling and simulation have focused on micro level components rather than on macro level integration of these components. However, with the advent of large-scale systems such as extended enterprises and distributed mission training, it is necessary to develop approaches for designing scalable Modeling and Simulation system architectures, including process specifications, linguistic support, granularity, and levels of abstraction to support system architecture design. This effort includes modularization, interconnectivity, and integration platforms as well as the standardization of application programs, automatic installation of modules, and verification. Metrics for such designs include robustness, reliability, flexibility, and the ability of the system to adapt dynamically to changing conditions when simulating large numbers of agents, determining an adequate level of fidelity for individual agents' behavior, validating agent-based models, and avoiding ad hoc assumptions during model development.

Semantic Consistency. Semantic Consistency, also known as substantive interoperability, refers to consistent phenomenological representations of modeling of real-world systems and processes among interacting distributed simulations. For example, two combat simulations must have consistent models of inter-visibility or they will be unable to interoperate meaningfully in a distributed simulation. Research into semantic consistency and a general mathematical language for expressing models are recommended. Dealing with Complexity and Errors Abstraction is the process of extracting a relatively sparse set of entities and relationships from a complex reality to produce a valid simplification of that reality. Abstraction is a general process; it includes simplification approaches such as aggregation, omission of variables and interactions, linearization, replacing stochastic processes by deterministic ones (and conversely), and changing the formalism in which models are expressed.

Complexity. The complexity of a model is measured in terms of the time and space required to execute it as a simulation. The more

NOTES

detail included in a model, the greater the resources required of the development team to build it and to execute it as a simulation once it is built. Validity is preserved through appropriate morphism mappings at desired levels of specification. Thus, abstraction methods, such as aggregation, will be framed in terms of their ability to reduce the complexity of a model while retaining its validity relative to the given modeling objectives inevitable resource constraints require working with models at various levels of abstraction. The complexity of a model depends on the level of detail, which in turn depends on the size/resolution product. The size/resolution product reflects the fact that increasing the size, or number of components, and resolution, or number of states per component, leads to increasing complexity. Since complexity depends on the size/resolution product, complexity can be reduced by reducing the size of the model or its resolution or both.

Several new approaches to modeling complexity are being developed. One of them is the notion of coordinated families of simulations at different levels of resolution. This approach presupposes the existence of effective ways to develop and correlate the underlying abstractions. A second approach, exploratory analysis, attempts to overcome computational complexity by addressing the issue of optimization, or searching through large spaces of alternatives for best solutions to a problem. This approach uses low-resolution models with a wide scope intended to capture the main features of an overall system or scenario. The approach seeks to exploit the reduction in the large space of alternatives that low-resolution, or highly abstracted model structures, may provide. A third approach fundamentally reconsiders the issue of optimization as a search for the best among many alternatives. The fast, frugal, and accurate (FFA) perspective on real-world intelligence provides a framework for insight into this issue. FFA is taken from the domain of human decision making in which full optimization is associated with unbounded rationality. This perspective recognizes that the real world is a threatening environment in which knowledge is limited, computational resources are bounded, and little time is available for sophisticated reasoning. Simple building blocks that steer attention to informative cues, terminate search processing, and make final decisions can be put together to form classes of heuristics that perform atleast as well as more complex algorithms.

Fundamental Limits of Modeling and Computation. In order to satisfy the needs of simulation for increasingly complex systems and processes, an integration of the statistics-oriented approach and simulation research must be emphasized by the academic community and the computer-science-oriented approach in acquisition and manufacturing. The statistics-oriented approach deals with prediction and management of uncertainty, whereas the computer-science-oriented approach deals with interoperability, reusability, integration, distributed operation,

and human/machine interfaces. The computer-science-oriented approach is necessary for the future operational success of defense acquisition and commercial manufacturing, but as processes and systems become increasingly complex, estimation and management of uncertainties will become increasingly important. Some fundamental limitations in computation in dealing with complex systems must be recognized. The performance of any future complex system will be unavoidably stated in probabilistic terms. A suite of software and a collection of databases may be technically interoperable and can be used to calculate system performance under a given set of operating environments, but there is no way that these tools can estimate the percentage of time that the system will perform satisfactorily under different circumstances, what the expected performance will be under uncertainty, or what the confidence level of the estimate is.

Performance Estimate. In addition, in order to improve the system performance estimate by adjusting or tuning various parameters in different phases of the acquisition process, dimensionality, or combinatorial explosion, must be dealt with. The first fundamental limitation in computation states that each system performance evaluation via simulation is time consuming. The second limitation states that a very large number of such evaluations may be necessary. These difficulties are multiplicative. Finally, there is a third limitation is "No Free Lunch Theorem". Without specific structural assumptions, there exists no optimization or search algorithm that can perform better on the average than blind search in dealing with the first and second limitation. These three limitations are fundamental limits on computation in dealing with complex systems. No amount of theoretical, hardware, or software advances can overcome them. Consequently, a strategic redirection is called for in dealing with them. Several emerging trends that directly or indirectly address the problem of system engineering of complex systems are outlined below. One or more of these topics may blossom into proven tools for dealing with the preceding difficulties and enable a more quantitative and optimizing approach.

Therefore, to deal with the large search spaces imposed by the second and third computational limitations the structure of specific problems must be learned along the way. A number of automated learning theories currently in trend in artificial intelligence research, such as knowledge discovery, data mining, Bayesian networks, and Tabu search, may be significant for developing modeling capabilities. Tabu search is a heuristic technique for search in combinatorial optimization problems.

Errors. Errors in Distributed Simulations fix resources and a model complexity that exceeds these resources, a trade-off must be made between size and resolution. If some aspects of a system are represented very accurately, only a few components will be representable. Alternatively,

NOTES

a comprehensive view of the entire system can be provided, but only at a low resolution. Such resolution may introduce errors that may pose particular problems in distributed simulations. In such complex, networked systems of models, owing to low resolution each model will typically be in error to some degree. Therefore, it is natural to expect that in a complex system of many linked models, even if individual inaccuracies are small, such errors can accumulate, propagate, and reinforce each other, rendering the behavior of the aggregate significantly different from the behavior of the real system. Error propagation in distributed simulations plays an important role in verification, validation, and accreditation, and therefore is an important area of research that needs to be strengthened. In the current state of the art, it is possible to suggest that such error propagation may, or may not be, a significant issue in distributed simulations. On the one hand, modeling errors in complex systems can be like noises that are more or less statistically independent. The cumulative effect of many independent errors behaves according to the central limit theorem and decrease with increasing complexity under some reasonable assumptions. A simple case is the law of large numbers, which improves accuracy by averaging many measurements. A second mitigating factor is the theory of ordinal optimization, mentioned above. Research here has shown that for the purpose of comparison (for example, which is better?), very crude models are quite sufficient.

Model Correctness. Model correctness is the fundamental requirement of ensuring that the predictions of a simulation model can be relied upon. The correctness of simulation requires the development of accurate and reliable models of real-world systems. A prerequisite to this is an understanding of the real-world systems and objects to be modeled, their contextual domains, and the phenomenology of their operations and interactions, all at a level of detail sufficient to justify the model. Once the models have been implemented as simulations, their correctness must be rigorously evaluated. Domain Knowledge improved understanding of the real-world basis for models is needed in the areas of phenomenology of warfare, physics-based modeling, and human behavior modeling. Phenomenology of Warfare the military domain is of special importance because it is the primary focus of SBA and because it is the domain in which human lives are most likely to be risked on the basis of decisions made using modeling and simulation.

Human Behavior Modeling. Human Behavior Modeling Computer generated forces are often used in training simulations to provide both opposing forces and supplemental friendly forces for human participants in a simulation. They are also often used to generate all of the entities in battlefield simulations being used for non-training purposes, such as analysis and experimentation. Automated or semi-automated entities are created, and their behavior is controlled by the computer system, perhaps assisted by a human operator, rather than by human participants in a simulator. These automated behaviors

are produced by algorithms based on models of human behavior. The reliability of the results depends on the validity of the behavior-generation methods. While current behavior-generation methods are reasonably effective at producing behavior that is in accordance with straightforward strategic policy, they fall far short of producing realistically human behavior with all its unpredictability and sophistication. Several studies have concluded that a need exists for improvement in human behavior modeling.

1.6 MODEL VERIFICATION AND VALIDATION

A key area of the model development process in the simulation area is the development of verification and validation stages in the process. Management scientists have had very little to say about how one goes about "verifying" a simulation model or the data generated that the modeling process has the following five steps:

1. **Systems Analysis:** The study of a system in order to ascertain its salient elements and to outline their interactions and behavior mechanisms;
2. **System Synthesis:** The construction of a complete, logical structure in order to provide a reasonable symbolic mimicry, or model of the system's elements and interactions, including the determination and collection of data required to support the model's structure;
3. **Verification:** Verification is concerned with building the model right. It is utilized in the comparison of the conceptual model to the computer representation that implements that conception. Simulation models can be run interactively or in batch mode. Interactive runs are of use in checking out (verifying) model's logic during model-building and in troubleshooting a model when execution errors occur. Batch mode is then used to make production runs. Interactive runs put a magnifying glass on a simulation model while it executes. The modeler can follow the active entity step by step and display the current and future events lists and the delay and user-managed lists as well as other aspects of the model. These activities yield valuable insights into model behavior for the modeler who knows the underlying concepts. Without such knowledge, the modeler might not take full advantage of the interactive tools provided by the software or, worse yet, might even avoid using the tools.
4. **Validation:** Validation is concerned with building the right model. It is utilized to determine that a model is an accurate representation of the real system. The validation is process of

NOTES

comparison of responses emanating from the verified model with available information regarding the corresponding behavior of the simulated system; and

5. **Model analysis :** The process of contrasting of model responses under alternative environmental specifications (or input conditions).

The general level of agreement on the definition of verification and validation should not however be taken as evidence to suggest that this part of the model development process is either simple or straightforward.

Case Study : Hospital Modeling

Campbelltown Public Hospital (CPH) is a modern and expanding hospital, in a fast-growing part of Australia. The hospital has 210 beds currently in use and offers a wide variety of services which include Medical, Surgical, Maternity, Pediatric, Intensive Care, Coronary Care, Orthopedics etc. These services are supported by modern Pathology and X-Ray services, Operating Theatres and an Accident and Emergency Department. CPH is an associate teaching hospital and is expected to have a capacity of about 400 beds by the turn of the century.

Identifying Goals. The public health system in Australia, and New South Wales is under increasing pressure to increase levels of service, with levels of financial support which do not match the levels of increasing demand on the service. One such measure is the service time for clients categorized according to the level and urgency of attention required. Within the emergency treatment ward, the patients enter the ward and are assigned a category by a triage nurse. This category is used to prioritize the patient in most subsequent operations in the system. Funding for the hospital is influenced by the promptness of service for each of the five categories of the patients. This model is designed to explore the impact of different management strategies on the level of service by category of patients. The initial range of management strategies assumes relatively constant levels of human resources in the department. The strategies are mainly directed at rescheduling of these resources.

The Process of Developing and Using the Model

The development model of CPH's emerging world is shown in Fig. 1.2.

Process Description. The model of this process has been constructed using one entity, *i.e.,* the patient. The patient is characterized with triage category, and with age. This characterizes the major attributes of the patients given the scope of this model. The triage category is

determined either by the triage nurse at the triage work centre or prior to arrival at the hospital, depending on the seriousness of the patient's condition. The age of the patient is used to categorize the patient as pediatric or adult. This requires different policies to be exercised at different positions in the model. The model contains six types of work center; Waiting Room (1), Clerical office (1), Traige (1), Cubicles (5), Resuscitation (2) Observation (6) and Overflow (7). Patients can move from various work centers to others depending on the state of the work centre, availability of resource, category of injury and age of patient. The number of alternative paths through the set of workstations is too numerous to enunciate in this paper. The model contains six types of resource. All resources in this model are people. Resources are clerks, triage nurse, senior doctor, doctor, registered nurse team leader and registered nurse. The number of resources available at any particular time in the execution of the model is variable, depending on the shift structure chosen for that model. Complexity in this model exists in the policies which are used to guide prioritization of patients and allocation of resources to patients in work centers.

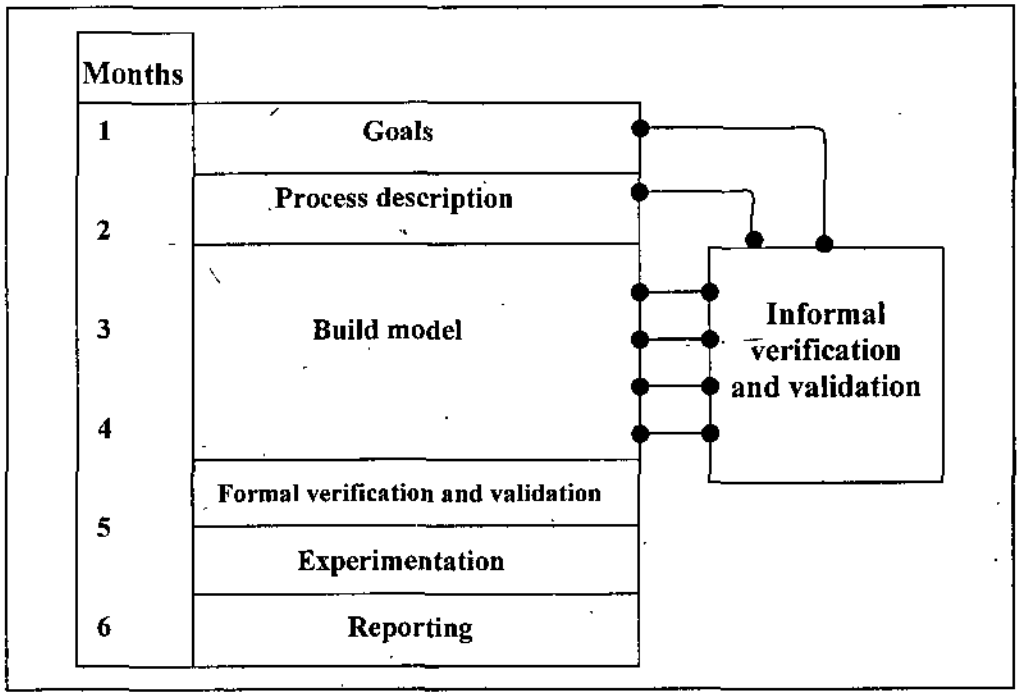

Fig. 1.2 Model development process

Building the Model. The process of developing and building the model of the emergency ward was a team effort which included an experience staff member from the hospital. Throughout the model building phase successive models have been placed before other stakeholders within the reference system. In general the concerned hospital staff has been asked to comment, or react to the animated interface. At times, they have been asked to comment on flowcharts and verbal descriptions of policy rules. The use of flowcharts was a

NOTES

key aspect of the early stages of developing the model as it enabled the system experts to articulate the system in terms which they understood, and which were readily able to be translated into discrete event simulation concepts. During the model building process validation and verification techniques were used at many points.

Verifying the Model. As the model has been developed, the builder has subjected the model to special input testing. The most challenging aspect of verification for this model is in the pre-emption strategies which take place for various resource/entity combinations. Verification has been conducted by close observation of the animated interface during extended runs of the model. Conflicts or inconsistencies in the behavior of the icons have led to examination of the code and to the correction of incorrect code and, more commonly, to the inclusion of further policy rules.

Several times during the model building phases the model was shown to system experts. They were asked to observe the flow of entities and resources throughout the workstations during extended runs of the model. This did not lead to significant levels of feedback which might have contributed to model verification. This was despite the presence of errors in the code, errors which led to visible inconsistent behavior of the entities and resources. Inconsistencies which were noted by system experts were often a result of verbal description of inconsistent behavior of the model. These inconsistencies were noted, not by design, but by chance. In general the discussions were prompted by problems with the model which caused the model builder to provide a general description of policy and rules for that particular part of the system. It was often in providing background to what the model was doing that these inconsistencies were noted. As the model building stage neared completion a structured walkthrough of the code was conducted. This led to the identification of some code which was redundant and therefore confusing, but not to errors in the execution of policy.

Validation of Model. Two broad techniques were used to validate the model. The first of these, and the most extensively used, was visualization and animation. A typical and repeating pattern of a typical day's arrivals for patients was used as the basic model platform. The model was run for extended periods and system experts and members of the modeling team observed the pre-emption behaviors of entities and resources. They have also been asked to carefully observe the behavior of queues in front of the various workstations.

The model generally satisfied these groups with respect to its broad, informal dynamic behavior. The number of people in the waiting room generally conformed to the expectation of the system experts. Queues were seen to be sensible, given the arrivals of different categories of patients. To facilitate this stage of validation, patients with different

categories were represented with different colored icons, and substantial reporting of quantitative was appended to the animation interface. This provided observers of the model with a detailed narrative of the state of each resource, entity and location work centre. The second validation strategy technique was a graphical presentation of the models behavior. The goal of the model was to provide decision makers with a tool to examine strategies to achieve better service provision, particularly for patients with non-critical injuries. The model therefore had to be valid within the domain of these goals. Statistical data was collected for extended periods of hospital history, and this data was analyzed by patient category. The model was instrumented in order to provide data in the same format. A simple comparison of frequencies of service levels by patient category was used to establish validity levels for the model. A sample of this report is shown in Fig. 1.3. The consistency between hospital historical data and model outputs at this stage has provided the model development team and the client team with the confidence to provide resources for further development of the model.

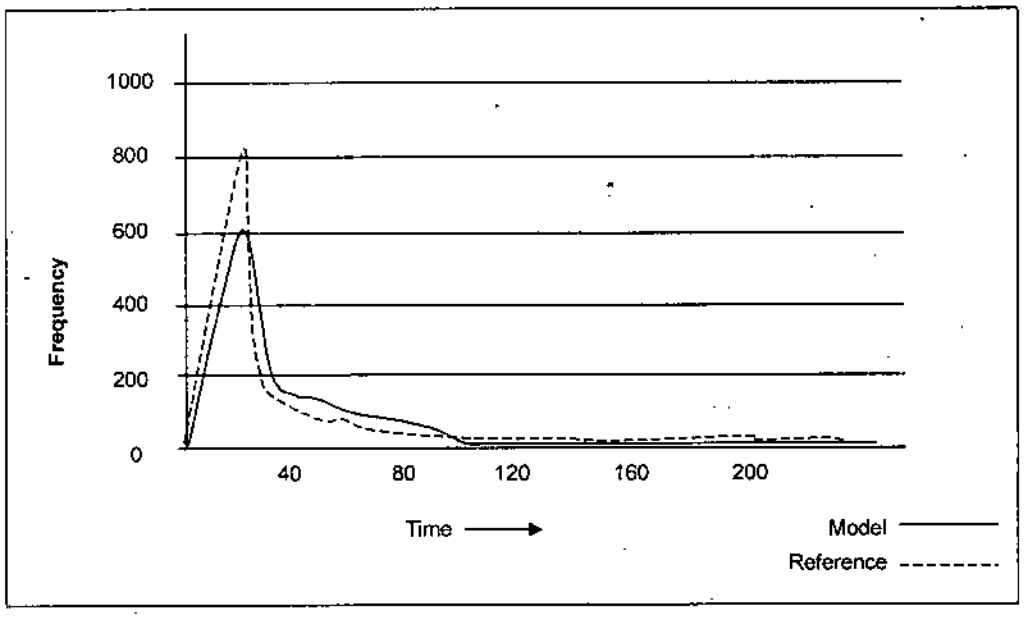

Fig. 1.3 Model and reference waiting times in CPH

Model Behavior. The model is currently in the analysis stage. Hospital management has accepted the validity of the model, and has proposed a number of scenarios which the modeling team are currently experimenting with. The scenarios are directed mainly at resource allocation options. One scenario, however illustrates the benefit of animated discrete event simulation. The potential to introduce a paramedic for patients with low priority had been dismissed during previous discussions of emergency ward management. The model has enabled the client team to quantify the benefits which could flow from this strategy and thus it has enabled them to give this strategy a more considered response.

NOTES

1.7 TYPES OF MODELS

There are following types of models. We will discuss about them as follows.

Continuous-time Models

Continuous-time models are the models used for continuous time simulations. In these cases, state or the variables of the system change with time. Continuous-time models are created and recognized by the property 'Ts' = 0. You can create and analyze all objects as continuous-time models by setting 'Ts' equal to zero at the time of creation, as in following MATLAB statements:

```
m = idpoly(1, [0 2 1], 1, 1, [1 3 4], 'Ts', 0)
```

for the model $y = \frac{2s+1}{s^2+3s+4}u + e$

all model characteristics are then computed and graphed for the continuous-time representation. Time and frequency scales are determined based on the dynamics of the system (the pole/zero locations). For simulation and prediction, the continuous-time models are first converted to discrete time, using the sampling interval and inter sample behavior of the data.

Estimating Continuous-time Models

The estimation routines support the estimation of continuous-time state-space models in several different ways. The major reason for identifying continuous-time models is to secure a particular structure of the continuous-time state-space matrices. This would typically reflect a physical interpretation or some gray-box modeling work done, as for the process models, described.

Transformations

Transformations between continuous-time and discrete-time model representations are performed by `c2d` and `d2c`. Note that it is not sufficient just to assign a new value of `Ts` to the model object. The corresponding uncertainty measure (the estimated covariance matrix of the internal parameters) is also transformed in most cases. The syntax is :

`modc = d2c(modd)` for discrete to continuous

`modd = c2d(mc,T)` for continuous to discrete

The transformation `c2d` also offers an optional output argument that describes how the initial state should be transformed.

If the discrete-time model has some pure time delays, the default command removes them before forming the continuous-time model, and appends them using the property `InputDelay` in model `modc`. This property is used to add appropriate phase lag and shift the data whenever the model is used. `d2c` also offers an option to approximate the dead time by a finite dimensional system. Note that the disturbance properties are translated by the somewhat questionable formula.

The covariance matrix is translated by the Gauss approximation formula using numerical derivatives. Here is an example that compares the Bode plots of an estimated model and its continuous-time counterpart.

- `m = armax(Data,[2 3 1 2]);`
- `mc = d2c(m); bode(m,mc)`

The transformations between discrete and continuous time depend on the inter sample behavior of the input. The formulas are different if the input is assumed to be piecewise constant or piecewise linear between samples (`'zoh'` or `'foh'`). For estimated discrete-time models, the input properties of the estimation data are used for this purpose, by default. To override this, add an extra argument, as described in the reference pages for `c2d` and `d2c`.

Discrete-time Model

Such models are used for discrete time simulations. Discrete time simulation is commonly used by the operations research work to study large, complex systems which do not lend themselves to a conventional analytic approach. A well known example of discrete time simulations is inventory model simulation. Airports, telephone exchanges, production line, stock of goods are some other examples of discrete time models.

This kind of model setup allows us to directly assess the structural inter-dependencies among the shocks to returns and the two different volatility components. The model estimates suggest that the leverage effect, or asymmetry between returns and volatility, works primarily through the continuous volatility component. The excellent fit of the model makes it an ideal candidate for an easy-to-implement auxiliary model in the context of indirect estimation of empirically more realistic continuous-time jump diffusion effectively incorporating the relevant information in the high-frequency data.

Continuous and Discrete-event Models

System or models in which changes are predominantly discontinuous are Discrete Simulation Models. Discrete-event simulation concerns the modeling of a system as it evolves over time by a representation in which state variables change instantaneously at separate points in

NOTES

time. The amount of data that must be stored and manipulated for most real-world systems dictates that discrete-event simulation is done on a digital computer.

Continuous simulation concerns the modeling over time of a system by representation in which the state variables change continuously with respect to time. Continuous simulation models involve differential equations that give relationships of the rates for change of the state variables with time. If the differential equations are particularly simple, they can be solved analytically to give the values of the state variables for all values of time as a function of the values of the state variable at time 0. For most continuous models analytic solutions are not possible and numerical analysis techniques as Runge-Kutta integration are used to integrate the differential equations numerically, given specific values for the state variables at time 0.

Discrete event models use discrete events as a set of circumstances that cause instantaneous changes in one or more system state descriptions. Since some systems are neither completely discrete nor completely continuous, the need may arise to construct a model with aspects of both discrete-event and continuous simulation resulting in a combined discrete-continuous simulation. The three fundamental types of interactions that can occur between discretely changing and continuously changing state variables:

- A discrete event may cause a discrete change in the value of continuous state variables.
- A discrete event may cause the relationship governing a continuous state variable to change at a particular time.

A continuous state variable achieving a threshold value may cause a discrete event to occur or to be scheduled.

Deterministic and Stochastic Models

Deterministic models produce deterministic results. If a simulation model does not contain any probabilistic or random components, it is called deterministic. In deterministic models, the output is 'determined' once the set of input quantities and relationships in the model have been specified. It takes a lot of compute time to evaluate what this output can be. When a system is modeled with at least some random input components then it is stochastic simulation model. The output produced by stochastic model is random. The outputs only estimate the true characteristics of model. This is the main disadvantage of stochastic simulation model. Stochastic or probabilistic models are subject to random effects:

- Typically, they have one or more random inputs *e.g.*, arrival of customers, service time etc.

NOTES

- Outputs from stochastic models are "estimates" of the true characteristics of the system.
- Need to repeat experiments number of times.
- Need to have confidence in the results.

Static and Dynamic Models

Static Models—If the system state is independent of time. A static simulation model is a representation of a system at a particular time. These models may be used to represent a system in which time simply plays no role, for example, Monte Carlo Models.

Dynamic Models—A model is called dynamic model if the system state changes with time. A dynamic simulation model represents a system as it evolves over time, for example, Conveyor system in a factory.

Linear and Non-linear Models

- ***Linear Models***—The model in which output is a linear function of input parameters
- ***Non-linear Model***—A model can be nonlinear in its parameters, nonlinear in its observed variables, or nonlinear in both its parameters and variables. Nonlinear in the parameters means that the mathematical relationship between the variables and parameters is not required to have a linear form. A linear model is a special case of a nonlinear model.

1.8 THE MODELING

The way of creating or developing a model is called modeling. Modeling means the process of organizing knowledge about a given system. A model is a pattern, plan, representation, or description designed to show the structure or workings of an object, system, or concept.

Problem Formulation: Problem formulation is concerned with:

- Identify controllable and uncontrollable inputs.
- Identify constraints on the decision variables.
- Define measure of system performance and an objective function.
- Develop a preliminary model structure to interrelate the inputs and the measure of performance.

Data Collection and Analysis: Regardless of the method used to collect the data, it is concerned with the decision of how much to collect be a trade-off between cost and accuracy.

NOTES

Simulation Model Development: Acquiring sufficient understanding of the system to develop an appropriate conceptual, logical and then simulation model is one of the most difficult tasks in simulation analysis.

Model Calibration: The process of parameter estimation for a model. Calibration is a tuning of existing parameters and usually does not involve the introduction of new ones, changing the model structure. In the context of optimization, calibration is an optimization procedure involved in system identification or during experimental design.

Input and Output Analysis: Discrete-event simulation models typically have stochastic components that mimic the probabilistic nature of the system under consideration. Successful input modeling requires a close match between the input model and the true underlying probabilistic mechanism associated with the system. The input data analysis is to model an element (like arrival process, service times, etc.) in a discrete-event simulation given a data set collected on the element of interest. This stage performs intensive error checking on the input data, including external, policy, random and deterministic variables. System simulation experiment is to learn about its behavior. Careful planning, or designing, of simulation experiments is generally a great help, saving time and effort by providing efficient ways to estimate the effects of changes in the model's inputs on its outputs. Statistical experimental-design methods are mostly used in the context of simulation experiments.

Sensitivity Estimation: Users must be provided with affordable techniques for sensitivity analysis if they are to understand which relationships are meaningful in complicated models.

Optimization: Traditional optimization techniques require gradient estimation. As with sensitivity analysis, the current approach for optimization requires intensive simulation to construct an approximate surface response function.

Stochastic Modeling

Stochastic means being or having a random variable. A stochastic model is a tool for estimating probability distributions of potential outcomes by allowing for random variation in one or more inputs over time. The random variation is usually based on fluctuations observed in historical data for a selected period using standard time-series techniques. Distributions of potential outcomes are derived from a large number of simulations (stochastic projections) which reflect the random variation in the input provided to the system.

A stochastic model can be used to set up a projection model which looks at a single policy, an entire portfolio or an entire company. But rather than setting investment returns according to their most likely estimate, for example, the model uses random variations to look at what investment conditions might be like.

Based on a set of random outcomes, the experience of the policy/ portfolio/organization is projected, and the outcome is noted. Then this is done again with a new set of random variables. In fact, this process is repeated thousands of times. At the end, a distribution of outcomes is available which shows not only what the most likely estimate, but what ranges are reasonable too. Stochastic modeling builds volatility and variability (randomness) into the simulation and therefore provides a more accurate representation of real life.

Stochastic Processes

A stochastic process is a probabilistic model of a system that evolves randomly in time and space. Formally, a stochastic process is a collection of random variables $\{X(t), t \in T\}$ all defined on a common probability space. The $X(t)$ is the state while time t is the index that is a member of set T.

Examples are the delay $\{D(i), i = 1, 2, ...\}$ of the i^{th} customer and number of customers $\{Q(t), t \geq 0\}$ in the queue at time t in an M/M/1 queue. In the first example, we have a discrete-time, continuous state, while in the second example the state is discrete and time in continuous.

The man made systems have mostly discrete state. Monte Carlo simulation deals with discrete time while in discrete even system simulation the time dimension is continuous.

Simulation Output Data and Stochastic Processes. To perform statistical analysis of the simulation output we need to establish some conditions, *e.g.,* output data must be a covariance stationary process (*e.g., the data collected over n* simulation runs).

Stationary Process (strictly stationary). A stationary stochastic process is a stochastic process with the property that the joint distribution all vectors of h dimension remain the same for any fixed h.

First Order Stationary. A stochastic process is a first order stationary if expected of $X(t)$ remains the same for all t. For example in economic time series, a process is first order stationary when we remove any kinds of trend by some mechanisms such as differencing.

Second Order Stationary. A stochastic process is a second order stationary if it is first order stationary and covariance between $X(t)$ and $X(s)$ is function of t's only. Again, a process is second order stationary when we stabilize also its variance by some kind of transformations such as taking square root. Clearly, a stationary process is a second order stationary, however the reverse may not hold. In simulation output statistical analysis we are satisfied if the output is covariance stationary.

Covariance Stationary. A covariance stationary process is a stochastic process $\{X(t), t \leq T\}$ having finite second moments, *i.e.,* expected of $[X(t)]^2$ be finite. Clearly, any stationary process with finite second moment is covariance stationary. A stationary process may have no

NOTES

finite moment whatsoever. Since a Gaussian process needs a mean and covariance matrix only, it is stationary (strictly) if it is covariance stationary.

Two Contrasting Stationary Process. Consider the following two extreme stochastic processes:

- A sequence Y_0, Y_1,....., of independent identically distributed, random-value sequence is a stationary process, if its common distribution has a finite variance then the process is covariance stationary.
- Let Z be a single random variable with known distribution function, and set $Z_0 = Z_1 = ...Z_i$. Note that in a realization of this process, the first element, Z_0, may be random but after that there is no randomness. The process $\{Z_i, i = 0, 1, 2, ..\}$ is stationary if Z has a finite variance.

Output data in simulation lies between these two types of process. Simulation outputs are identical and mildly correlated. It depends on a queueing system how large is the traffic intensity. An example could be the delay process of the customers in a queueing system.

SUMMARY

- In recent times, industry in general has begun to accept that the engineering of systems, both large and small, can lead to unpredictable behavior and the emergence of unforeseen system characteristics or emergent properties. Decisions made at the beginning of a project whose consequences are not clearly understood can have enormous implications later in the life of a system, and it is the task of the modern systems engineer to explore these issues and make critical decisions. There is no method which guarantees that decisions made today will still be valid when a system goes into service years or decades after it is first conceived but there are techniques to support the process of systems engineering. Examples include the use of soft systems methodology, the Unified Modeling Language etc., each of which are currently being explored, evaluated and developed to support the engineering decision making process.
- Modeling and simulation is the field closely related with system engineering. Systems engineering often involves the modeling or simulation of some aspects of the proposed system in order to validate assumptions or explore theories. For example, highly complex systems such as aircraft are usually modeled and simulated before flight. In this way the initial aero-elastic engineering and control equations can be drafted and improved upon before any physical system is ever constructed or developed. Since aircraft are often very expensive, this reduces the expense and difficulty of debugging the controls and reduces the risk of crashing real aircraft. Careful initial testing and flight envelope expansion are typically still

NOTES

required to reach acceptable levels of safety and performance in advanced aircraft.

- The role of the system engineer is especially important when systems must have especially predictable and reliable behavior. For example, medical machinery, power plants, and spacecraft usually consist of many individually engineered and manufactured parts, by different companies. System engineering provides the assurance that normal operations, including parts failures, will not provide a hazard for the user or anyone else in the community. The application of systems engineering processes may also result in significant cost savings, as well as providing a reasonable assurance of the eventual success of the project.
- Systems Engineering (SE) is an interdisciplinary approach and means for enabling the realization and deployment of successful systems. It can be viewed as the application of engineering techniques to the engineering of systems, as well as the application of a systems approach to engineering efforts. Systems engineering integrates other disciplines and specialty groups into a team effort, forming a structured development process that proceeds from concept to production to operation and disposal. Systems engineering considers both the business and the technical needs of all customers, with the goal of providing a quality product that meets the user needs.
- System development often requires contribution from diverse technical disciplines. Each of these disciplines is normally focused on their own particular contribution to the system (for example, jet engine designers would not be focused on the aircraft's hydraulic subsystem). System engineering's advantage point is a holistic perspective of the system and from this perspective integrates all of these technical efforts to ensure that their various subsystems work with one another. By providing a systems view of the development effort, System Engineering helps meld all the technical contributors into a unified team effort, forming a structured development process that proceeds from concept to production to operation and, in some cases, through to termination and disposal. System Engineering is usually directly responsible for any engineering function that is not deemed sufficiently necessary on a project to require a full-time, specialist engineer, although consultants may be enlisted as needed.
- Ideally, Systems Engineering considers both the business and the technical needs of all customers with the goal of providing a quality product that meets the user needs. However, the reality in any very large project is often that user needs exceed what the sponsor is willing to pay for; and the schedule to satisfy those needs generally exceeds what either of them is willing to live with. As a result, 'satisfaction of all technical requirements' is subject to the usual constraints of cost, schedule, and producibility.
- Taking an interdisciplinary approach to engineering systems is inherently complex, since the behavior of and interaction among system components are not always well defined or understood Defining and characterizing

NOTES

such systems and subsystems, and the interactions among them, is the primary aim of systems engineering. On very large programs, a systems architect may be designated to serve as an interface between the user/sponsor and systems engineer.

- There are several methods and tools that are frequently used by systems engineers: requirements capture, systems architecture and design, functional analysis, interface design and specification, communications protocol design and specification, simulation and modeling, verification and validation/acceptance testing, Fault modeling.

GLOSSARY

- **State:** Collection of variables and their values necessary to characterize a system at a particular time.
- **Entity:** An object that passes through the system.
- **Scheduling:** Is the act assigning a new future event to an existing entity.
- **Distribution:** Is the mathematical law, which govern the probabilistic features of a random variable.
- **Random variate:** Is an artificially generated random variable.

REVIEW QUESTIONS

1. Explain the difference between mental model and mathematical model.
2. Explain the difference between static and dynamic model.
3. What do you understand by model validation, verification and calibration?
4. What do you mean by stochastic process and stochastic modeling?
5. Explain the principles used in modeling of a system.
6. Explain the difference between simulation and experiment. Explain the parameters used for experiments with a system.
7. Write short notes on:

   (*i*) Natural System (*ii*) Artificial System

   (*iii*) Hybrid System (*iv*) Model
8. What are different types of models? Give the difference between discrete and continuous models.

FURTHER READINGS

- '*Modeling and Simulation Concepts*', by Rajinder Kumar, Anil Kumar, Vikesh Kumar, Chandra Shekher Yadav. University Science Press.

UNIT II SIMULATION CONCEPTS

NOTES

★ STRUCTURE ★

2.0 LEARNING OBJECTIVES

After going through this unit, you will be able to:

- explain system simulation and why, when and how to simulate.
- describe basic nature and techniques of simulation.

NOTES

- comparise simulation and analytical methods.
- define types of simulation process, queueing system, and queueing models.
- illustrate the inventory system and its basic function.
- describe monte carlo simulation, distributed lag models and cobwed models.

2.1 INTRODUCTION

Simulation is one of the most powerful tools available to decision-makers responsible for the design and operation of complex processes and systems. It makes possible the study, analysis and evaluation of situations that would not be otherwise possible. In an increasingly competitive world, simulation has become an indispensable problem solving methodology for engineers, designers and managers. Simulation is the imitation of the operation of a real world process or system over time. Simulation involves the generation of an artificial history of the system, and the observation of that artificial history to draw inferences concerning the operating characteristics of the real system that is represented. Simulation is an indispensable problem-solving methodology for the solution of many real world problems. Simulation is used to describe and analyze the behavior of a system. Both existing and conceptual systems can be modeled with simulation.

Recent advances in modeling and simulation technologies make them increasingly appealing as a means of improving commercial manufacturing and defense acquisition. However, in order for these technologies to support the desired applications in commercial manufacturing and defense acquisition, additional research and development is needed. As challenge, the organizations involved in simulation and modeling are often asked to investigate emerging modeling and simulation technologies, efforts to develop them, and identify gaps that would have to be filled in order to make these emerging technologies a reality. The organizations rephrase this task and require determining those topics requiring research and development to be effectively used in commercial manufacturing and defense acquisition. The topics requiring research and development are identified by the committee on the basis of the challenges in the real world.

A framework for the modeling and simulation of hybrid analog/digital systems has long been needed. Today, the need to design mixed-signal chips to support the growth in wireless devices and next-generation automotive electronics has brought this problem to the foreground. Mixed-Signal Chips (MSCs) have been implemented as custom-application-specific integrated circuits (ASICs), but must now be mass-produced

for use in wireless technology. MSCs receive analog signals, process and manipulate them mainly in digital form, and reconvert them back to analog form. The challenge for systems design is the high level of functionality of an MSC.

NOTES

2.2 SIMULATION

Before we proceed further, it becomes necessary to define the term simulation in more suitable forms. We define simulation as the process of designing a model of a real system and conducting experiments with this model for the purpose of understanding the behavior of the system and evaluating various strategies for the operation of the system. Thus it is critical that the model be designed in such a way that the model behavior mimics the response behavior of the real system to events that take place over time. The terms model and system are key components of our definition of simulation. By a model we mean a representation of a group of objects or ideas in some form other than that of the entity itself. By a system we mean a group or collection of interrelated elements that cooperate to accomplish some stated objective. One of the real strengths of simulation is the fact that we can simulate systems that already exist as well as those that are capable of being brought into existence, *i.e.*, those in the preliminary or planning stage of development. Some other important definitions are:

Definition 1: Simulation is the use of system model that has the designed characteristics of reality in order to produce the essence of actual operation.

Definition 2: Simulation is representation of reality through the use of a model or other device, which will react in the same manner as reality under a given set of conditions.

Definition 3: A Simulation model may be defined as one which depicts the working of a large scale system of men, machine, materials and information operating over a period of time in a simulated environment of the actual real world conditions.

Definition 4: According to Churchman Simulation is defined as,

"A simulates B" is true if and only if:

(*a*) A and B are formal systems;

(*b*) B is taken to be a real systems;

(*c*) A is taken to be an approximation to the real system; and

(*d*) The rule of validity in A are non-error-free, otherwise A will become the real system.

NOTES

Thus, simulation is the manipulation of a model in such a way that it operates on time or space to compress it, thus enabling one to recognize the interactions that would not otherwise be apparent because of their separation in time or space.

Simulation in general is to pretend that one deals with a real thing while really working with an imitation. In operations research the imitation is a computer model of the simulated reality. A flight simulator on a personal computer is also a computer model of some aspects of the flight: it shows on the screen the controls and what the "pilot" is supposed to see from the cockpit.

To fly a simulator is safer and cheaper than the real airplane. For precisely this reason, models are used in industry commerce and military: it is very costly, dangerous and often impossible to make experiments with real systems. Provided that models are adequate descriptions of reality (they are valid), experimenting with them can save money, suffering and even time.

Modeling and Simulation is a discipline for developing a level of understanding of the interaction of the parts of a system, and of the system as a whole. The level of understanding, which may be developed via this discipline, is seldom achievable via any other discipline.

2.3 WHY GO FOR SIMULATION?

At the most general level, simulation is considered as a form of cognition. Cognition is the action or process of acquiring knowledge. There are three basic methods how to get information (knowledge) of objective reality: Experiment, Analysis, and Simulation. Let us take one practical example to demonstrate the nature of these three methods.

Experiment: Common example of experiment can seen as, take stopwatches and measure the time every train spends in the railway station. Count the trains, at the end sum all times and divide them by the number of trains. Experiment is always the most accurate method that should be used whenever it is feasible. Unfortunately very often the experiment is:

- Too dangerous (behavior of a nuclear reactor in critical situations, landing with a plane with one jet off, etc.)
- Too expensive (all cases that cause a damage, long experiments studying throughput of a data network using leased phone lines, etc.)
- Not possible at all if the system being investigated is not available (evaluation of more possible alternatives in the design stage.)

Analysis: Common example of analysis can be seen as, use a formula of the Queuing Theory to compute the average time spent in the

NOTES

system directly. To use a formula you will have to assume certain queuing model, which means a considerable simplification of the real system, and you will need some quantitative parameters.

Analysis (mostly mathematical) is typically based on strong assumptions that are rarely true in practical life. Another possible drawback of analytical methods is too complicated apparatus used and/or too time consuming computation. An example of this is analysis of Queuing Networks. On the other hand using formulae gives mostly fast results and it is possible to check a large number of alternatives by simply inserting different values of parameters to the formulae. Experimental methods are mostly much more time consuming. Another problem of analysis is availability of necessary parameters. Their exact measuring is also not necessarily feasible or it is impossible in the design stage. Using estimated data or data taken from other similar systems decreases credibility of results.

Simulation: Simulations may be performed manually. Most often, however, the system model is written either as a computer program or as some kind of input into simulator software. A simulation generally refers to a computerized version of the model, which is run over time to study the implications of the defined interactions. Simulations are generally iterative in there development. One develops a model, simulates it, learns from the simulation, revises the model, and continues the iterations until an adequate level of understanding is developed. Simulation is also an experimental method. Instead of experimenting with the real system the experiments are performed with the simulation model (whose design is thus the key point of simulation studies). Also simulation has many drawbacks. Here are the most important ones:

- Too demanding creation of simulation models.
- Programming simulation models in general languages (like Pascal, Fortran) is too difficult.
- There are efficient simulation languages but their mastering represents a big initial investment not always justified.
- There are simulation-tools based typically on some graphical technique that simplify or even automate creation of simulation models of certain class of systems.
- Only, limited knowledge of the system being simulated, and some quantitative parameters must be known.

These methods cannot be ranked because all of them have advantages and disadvantages. They can be compared only in the context of certain particular case taking into account various criteria.

Simulation could be much more flexible than analysis because simulation languages support generation of random numbers with practically any distribution. In the above example both random figures can be based on any distributions obtained experimentally. Nevertheless any distribution needs either several parameters (if it is a theoretical

NOTES

one) or directly the Distribution Function *i.e.,* the distribution is obtained by measuring. There can be also things in the system (typically in the design stage) that cannot be quantified and often it is necessary to accept the fact that there might be aspects we are not aware of at all, like to much time consuming computation. An example is analysis of large-scale systems with many components working in parallel. Because application of real parallelism is still not common, a program performed by a single processor simulates such systems. Parallel activities are then performed one at a time. The result of this is the fact that simulation could be much slower than the real time. In general 1 second of the model time takes 10 minutes of the CPU time. This of course disables application of simulation in real time control.

A general rule of thumb could be like that: "If the experiment is feasible, use it. It is always the best method because all aspects are taken into account. Even if other methods were used during the design stage, experiment can serve as a final evaluation of the system. If the experiment is not feasible try to find an appropriate analytical method. If it is not available, simulation can be used."

Simulation is not only the last option as it looks like in the above rule. Simulation can contribute very much to understanding of the system being analyzed not only by supplying answers to the questions that were originally given. Very often creation of the simulation model is the first occasion where certain things are taken into account. Specification of the simulated system can (and often it does) reveal errors or ambiguities in the system design. So simulation can help very much by avoiding future very expensive updating of the ready system. A simulation is an experiment performed on a model. A mathematical simulation is a coded description of an experiment with a reference to the model to which this experiment is to be applied. Simulation is imitation of the operation of a facility or process, usually using a computer.

2.4 WHEN TO USE SIMULATIONS?

Systems that change with time, such as a fuel filling center where vehicles come and go and involve randomness are required to simulate. Nobody can guess at exactly which time the next vehicle should arrive at the filling center, are good conditions for simulation. Modeling complex dynamic systems theoretically need too many simplifications and the emerging models may not be therefore valid. Simulation does not require so many simplifying assumptions, making it the only tool even in absence of randomness. Simulations are appropriate tool in following conditions:

- The knowledge gained in the designing of a model may be very helpful towards suggesting improvements in the system under investigation.

NOTES

- Simulation enables the study of, and experimentation with, the internal interaction of the complex systems or subsystems.
- To study the effect of alterations to an existing system. Simulation compares design alternatives for a system that does not exist.
- Informational, organizational, and environmental changes can be simulated, and the effect of these changes on the models behavior can be observed.
- By changing simulation inputs and observing the results of output, valuable insights may be obtained into which variable are most important and how these variables interact.
- Simulation can be used as instructive device to support analytic solution methodologies.
- Simulation can be used to experiment with the new designs or policies prior to implementation, so as to prepare for what may happen.
- Simulation can be used to verify analytical solution.
- Requirements can be determined by simulating different capabilities for a machine.
- Simulation can be used to study complex system, where analytic solutions are infeasible. Some modern systems are so complex that the interactions can be organized only through simulation.

A simulation is not appropriate in following conditions:

- If model assumptions are simple such that mathematical methods can be used to obtain exact answers (analytical solutions), or a common sense can be used to solve it.
- Simulation should not be used if the problem can be solved analytically.
- Simulation should not be used if it is easier to perform direct experiments.
- Simulation should not be used if the cost exceeds the savings.
- Simulation should not be used if the resources and time are not available.
- Simulation takes data, sometimes a large amount of data. If no data is available, not even estimates, simulation is not advised.
- Simulation should not be used if there is no ability to verify and validate the model.
- Simulation should not be used if the persons involved in simulation have unreasonable expectations.
- Simulation should not be used if the system behavior is too complex or cannot be defined.

NOTES

2.5 HOW TO SIMULATE?

System simulation is done to see its real life effect before its development. When the decision is made for simulation a proper study of the system is must. In approaching a system study, it is essential to consider first what mathematical techniques might be applied to drive analytical solution. It is matter of judgment to decide whether the degree of analysis is sufficient for simulation. When the decision is made for simulation in order to use a more realistic system, it is still important to limit the amount the detail in the model to the minimum level necessary. The step by step nature of the simulation technique means that the amount of computation increases very rapidly as the amount of detail increases.

Suppose we are interested in a vehicle fuel filling center. We may describe the behavior of this system graphically by plotting the number of vehicles in the filling center as the state of the system. Every time a vehicle arrives the graph increases by one unit while a departing vehicle causes the graph to drop one unit. This graph, could be obtained from observation of a real station, but could also be artificially constructed. Such artificial construction and the analysis of the resulting sample path (or more sample paths in more complex cases) consist of the simulation.

Final Experimental Design

If we have developed the model, verified its correctness, and validated its adequacy, we again need to consider the final strategic and tactical plans for the execution of the experiments. We must update project constraints on time (schedule) and costs to reflect current conditions. Even though we have exercised careful planning and budget control from the beginning of the project, we must now take a hard, realistic look at what resources remain and how best to use them. We will also have learned more about the system in the process of designing, building, verifying and validating the model which we will want to incorporate into the final plans. The design of a computer simulation experiment is essentially a plan for purchasing a quantity of information that costs more or less depending upon how it was acquired. Design profoundly affects the effective use of experimental resources because:

- The design of the experiments largely determines the form of statistical analysis that can be applied to the data.
- The success of the experiments in answering the desired questions is largely a function of choosing the right design.

Simulation experiments are expensive both in terms of the analyst's time and labor and in some cases, in terms of computer time. We must therefore carefully plan and design not only the model but also its use.

NOTES

Experimentation and Analysis

Next we come to the actual running of the experiments and the analysis of the results. We now have to deal with issues such as how long to run the model (*i.e.,* sample size), what to do about starting conditions, whether the output data are correlated, and what statistical tests are valid on the data. Before addressing these concerns, we must first ascertain whether the real system is terminating or non-terminating because this characteristic determines the running and analysis methods to be used. In a terminating system, the simulation ends when a critical event occurs. For example, a bank opens in the morning empty and idle. At the end of the day it is once again empty and idle. Another example would be a duel where one or both participants are killed or the weapons are empty. In other words, a system is considered to be terminating if the events driving the system naturally cease at some point in time. In a non-terminating system, no such critical event occurs and the system continues indefinitely (*e.g.,* a telephone exchange or a hospital). A second system characteristic of interest is whether the system is stationary or non-stationary. A system is stationary if the distribution of its response variable (and hence it is mean and variance) does not change over time. With such systems we are generally concerned with finding the steady state conditions, *i.e.,* the value which is the limit of the response variable if the length of the simulation went to infinity without termination. Whether the system is terminating or non-terminating, we must decide how long to run the simulation model *i.e.,* we must determine sample size. But first we must precisely define what constitutes a single sample. There are several possibilities:

1. Each transaction considered a separate sample. For example, turn-around time for each job or total time in the system for each customer.
2. A complete run of the model. This may entail considering the mean or average value of the response variable for the entire run as being a datum point. Multiple runs are referred to as replication.
3. A fixed time period in terms of simulated time. Thus a simulation may be run for n time periods, where a time period is an hour or a day or a month.
4. Transactions aggregated into groups of fixed size. For example, we might take the time in the system for each 10 jobs flowing through and then use the mean time of the group as a single datum point. This is usually referred to as batching.

If the system is a non-terminating, steady-state system we must be concerned with starting conditions, *i.e.,* the status of the system when we begin to gather statistics or data. If we have an empty and idle system *i.e.,* no customers present, we may not have typical steady

NOTES

state conditions. Therefore, we must either wait until the system reaches steady state before we begin to gather data (warm-up period), or we must start with more realistic starting conditions. Both of these approaches require that we be able to identify when the system has reached steady state.

Finally, most statistical tests require that the data points in the sample be independent *i.e.,* not correlated. Since many of the systems we model are queueing networks, they do not meet this condition because they are auto-correlated. Therefore, very often we must do something to assure that the data points are independent before we can proceed with the analysis.

Implementation and Documentation

At this point we have completed all the steps for the design, programming and running of the model as well as the analysis of the results. The final two elements that must be included in any simulation study are implementation and documentation. No simulation study can be considered successfully completed until its results have been understood, accepted and used. It is remarkable how often modelers will spend a great deal of time trying to find the most elegant and efficient ways to model a system and then throw together a report to the sponsor or user at the last minute. If the results are not used, the project was a failure. If the results are not clearly, concisely and convincingly presented, they will not be used. The presentation of the results of the study is a critical and important part of the study and must be as carefully planned as any other part of the project. Among the issues to be addressed in the documentation of the model and study are:

- Choosing an appropriate vocabulary (no technical jargon).
- Length and format of both written and verbal reports (short and concise).
- Timeliness
- Must address the issues that the sponsor or user consider important.

Paths to Failure

Not all simulation studies are unqualified successes. In fact, unfortunately, too many fail to deliver as promised. When we look at the reasons that projects fail, we find that it is usually traceable to the same reasons over and over. Most failures occur on early projects *i.e.,* the first or second project undertaken by an organization. Many inexperienced modelers bite off more than they can chew. This is not surprising since in most cases they have learned the science but not the art of simulation. This is why it is advisable to begin with small projects

that are not of critical significance to the parent organization. Almost all other failures can be traced to one of the following:

- Failure to define a clear and achievable goal.
- Inadequate planning and underestimating the resources needed.
- Inadequate user participation.
- Writing code too soon before the system is really understood.
- Inappropriate level of included detail (usually too much).
- Wrong mix of team skills.
- Lack of trust, confidence and backing by management.

Paths to Success

Just as we can learn from studying projects that fail, we can also learn from those that. Obviously the first thing we want to do is avoid the errors of those who fail. Thus we want to:

- Have clearly defined and achievable goals.
- Be sure we have adequate resources available to successfully complete the project on time.
- Have management's support and have it known to those who must cooperate with us in supplying information and data.
- Assure that we have all the necessary skills required available for the duration of the project.
- Be sure that there are adequate communication channels to the sponsor and end users.
- Have a clear understanding with the sponsor and end users as to the scope and goals of the project as well as schedules.
- Have good documentation of all planning and modeling efforts.

2.6 SIMULATION AND ANALYTICAL METHODS

Extensive development has been made out in both fields ever since the beginnings of quantitative science. Development of computing machines is evolved as substitute for labor of human computers. Computers lower cost of approximations as compared to analytical results. This can even lead to simply throwing it on the computer even when analytical results are obtainable; even, indeed, when analytical results are well-known in the literature.

Comparisons. In approaching a system study, it is essential to consider what mathematical techniques might be applied to derive analytical solutions. Mathematical techniques require that the models should be expressed in some particular format. The system must be approximated

NOTES

or abstracted in order to derive a model that fits the format of a mathematical technique. It is the matter of judgment to decide whether the degree of abstraction required to apply analytical methods is too difficult. To make this judgment consider all questions to be answered in system study and the level to which the accuracy of answers need to be known.

Comparison between Analytical and Simulation methods

Analytical Methods	*Simulation Methods*
Analytical Methods produce general solutions.	Simulation methods give specific solutions.
In analytical method, all the conditions involved to solve a problem are considered.	Each execution of a simulation tells only whether a particular set of conditions is successful or not.
Mathematical solution is preferable, when solution being sought is some maximizing condition.	Many simulation runs may be needed to find a maximum, and still undecided whether it is a local or global maximum.
Analytical method produces solutions in a single run.	To consider all conditions, the simulation process has to be repeated with many different conditions.
Many analytical results occur in the form of complex series that still require extensive evaluation.	Simulation is used to obtain results directly from a model with specific values.
Range of problems to be solved mathematically is limited.	Simulation is a powerful extension of mathematical solutions.
Analytical model of a system is fixed.	Simulation model of the desired system changes as analyst modeling of the system changes.
Analytical methods employ the dedicated reasoning of mathematics to solve the model.	Simulation models employ numerical methods; models are run rather than solved.
Analytical methods are expensive and time consuming.	Simulation gives results in few minutes at a very low cost.
Analytical model remains same; it does not depend on analyst.	Simulation model varies with experience of the analyst.

The step-by-step nature of the simulation technique means that the amount of computation increases very rapidly as the amount of details increases. The best way of using simulation is an extension of mathematical solution. This can be achieved at the cost of too much simplification. Simple limitations on the system can easily be removed by simulation.

When a solution of the problem is known, then also simulation provides a quick and more convenient way of deriving results.

2.7 BASIC NATURE OF SIMULATION

NOTES

The simulation techniques make no specific attempt to isolate the relationships between any particular variable, instead it explore the way in which all variables of the model change with time. For a better simulation, the relationship among the variable must be derived from the observations. Following are the main points regarding the nature of simulation:

Simulation imitates the operations of a facility or process, usually via computer:

- What is being simulated is the system?
- To study system, often make assumptions/approximations, both logical and mathematical operations, about how it works.

These assumptions form a model of the system to be simulated if model structure is simple enough, we can use mathematical methods to get exact information on questions of interest *i.e.*, analytical solution.

But most complex systems require models that are also complex (to be valid) it must be studied via simulation *i.e.*, evaluate model numerically and collect data to estimate model characteristics.

Example: Manufacturing Company considering extending its plant:

- Build it and see if it works out?
- Simulate current, expanded operations *i.e.*, we can also investigate many other issues along the way, quickly and cheaply.

Some (not all) application areas:

- Designing and analyzing manufacturing systems
- Evaluating military weapons systems or their logistics requirements
- Determining hardware requirements or protocols for communications networks
- Determining hardware and software requirements for a computer system
- Designing and operating transportation systems such as airports, freeways, ports, and subways
- Evaluating designs for service organizations such as call centers, hospitals, fast-food restaurants, and post offices
- Reengineering of business processes
- Determining ordering policies for an inventory system
- Analyzing financial or economic systems.

NOTES

Surveys of use of Operations Research / Modeling Simulation techniques:

- Simulation consistently ranked as one of the three most important techniques
- Simulation was second only to the broad category of "mathematical programming".

Impediments to acceptance, use of simulation:

- Models of large systems are usually very complex
- But now we have better modeling software *i.e.*, more general, flexible, but still (relatively) easy to use
- Can consume a lot of computer time
- But now have faster, bigger, cheaper hardware to allow for much better studies than just a few years ago and this trend will continue
- However, simulation will also continue to push the envelope on computing power in that we ask more and more of our simulation models
- Impression that simulation is "just programming"
- There is a lot more to a simulation study than just coding a model in some software and running it to get the answer.
- Require careful design and analysis of simulation models *i.e.*, simulation methodology.

2.8 THE SIMULATION PROCESS

The essence or purpose of simulation modeling is to help the ultimate decision-maker solve a problem. Therefore, to learn to be a good simulation modeler, you must merge good problem solving techniques with good software engineering practice. We can identify the following steps, which should be present in any simulation study:

1. **Problem Definition.** Clearly defining the goals of the study so that we know the purpose, *i.e.,* why are we studying this problem and what questions do we hope to answer?
2. **Project Planning.** It is assurance about sufficient and appropriate personnel, management support, computer hardware, and software resources to do the job.
3. **System Definition.** Determining the boundaries and restrictions to be used in defining the system (or process) and investigating how the system works.
4. **Conceptual Model Formulation.** It is developing a preliminary model either graphically (*e.g.,* block diagram or process flow chart), or in pseudo-code to define the components, descriptive variables, and interactions (logic) that constitute the system.

NOTES

5. **Preliminary Experimental Design.** Selecting the measures of effectiveness to be used, the factors to be varied, and the levels of those factors to be investigated, *i.e.*, what data need to be gathered from the model, in what form, and to what extent.
6. **Input Data Preparation.** Identifying and collecting the input data needed by the model.
7. **Model Translation.** It is formulation of the model in an appropriate simulation language.
8. **Verification and Validation.** Confirming that the model operates the way the analyst intended (debugging) and that the output of the model is believable and representative of the output of the real system.
9. **Final Experimental Design.** Designing an experiment that will yield the desired information and determining how each of the test runs specified in the experimental design is to be executed.
10. **Experimentation.** Executing the simulation to generate the desired data and to perform sensitivity analysis is experimentation.
11. **Analysis and Interpretation.** Drawing inferences from the data generated by the simulation runs.
12. **Implementation and Documentation.** It is the process of reporting the results, putting the results to use, recording the findings, and documenting the model and its use.

The activities and steps in modeling and simulation process are given by Fig. 2.1.

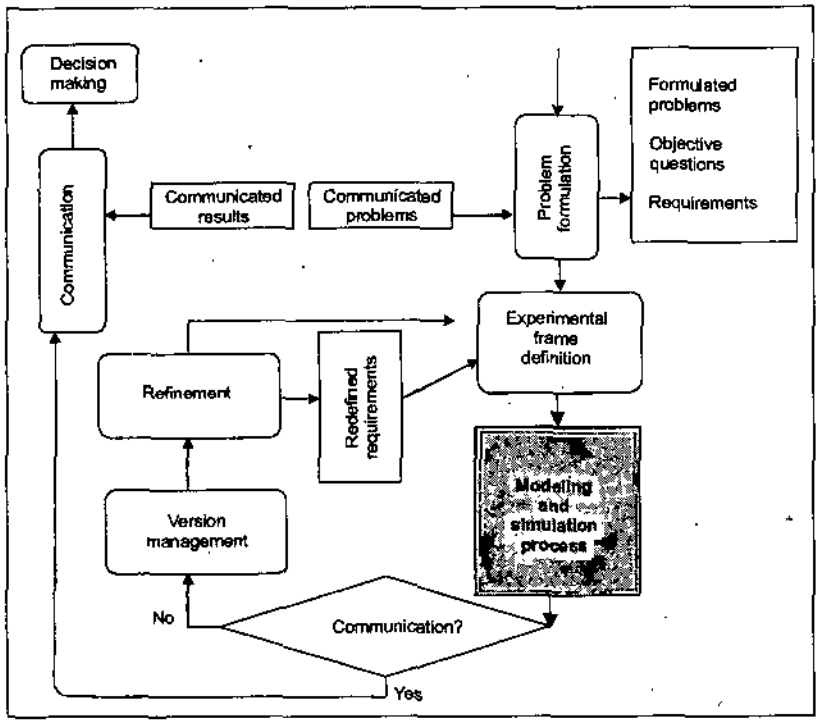

Fig. 2.1 Modeling and simulation process

NOTES

It is important for the inexperienced modeler to understand that the longer you wait to start step 7, the faster you will complete the model and the project—assuming of course, that you spend that time understanding the problem, designing the model and designing the experiments to be executed. Computer Scientists have devoted a great deal of effort to software engineering and have developed design and management methods aimed at yielding rapid programs while minimizing errors. One of the major ideas that have emerged from this effort is the validity of the "40-20-40 Rule". This rule states that 40 percent of the effort and time in a project should be devoted to Steps 1 through 6, 20 percent to step 7, and the remaining 40 percent to steps 8 through 12.

Problem Definition and Project Planning

Experience indicates that beginning a simulation project properly, may well make a critical difference between success and failure. We begin our analysis by gathering information about the problem. Usually, the project begins with the sponsor describing the situation to the analyst in vague and imprecise terms, such as costs are too high or, "too many jobs are late". We must consider the sponsor's problem description as a set of symptoms requiring diagnosis. The usual flow of events will be:

Diagnosis of symptoms ⇒ Problem definition ⇒ System definition ⇒ Model formulation.

It is important to remember that we do not model a system just for the sake of modeling it. We always model to solve a specific problem. Among the questions to be answered at the beginning of the study are:

- What is the goal of the study *i.e.,* what is the question to be answered or decision to be made?
- What information do we need to make a decision?
- What are the precise criteria we will use to make the decision?
- Who will make the decision?
- Who will be affected by the decision?
- How long do we have to provide an answer?

Once we have those answers we can begin to plan the project in detail. An important aspect of the planning phase is to assure that certain critical factors have been considered. Among these are:

- Do we have management support and has its support for the project been made known to all concerned parties?
- Do we have a competent project manager and team members with the necessary skills and knowledge available for sufficient time to successfully complete the project?
- Do we have sufficient time, computer hardware and software available to do the job?

- Have we established adequate communication channels so that sufficient information is available on project objectives, status, changes in user or client needs etc., to keep everyone (team members, management, and clients) fully informed as the project progresses?

Steps in a Simulation Study

Fig. 2.2 shows a set of steps to guide a model builder in a thorough and sound simulation study. The main steps are as following:

1. **Problem formulation.** Every simulation study begins with a statement of the problem. If the statement is provided by those that have the problem (client), the simulation analyst must take extreme care to insure that the problem is clearly understood. If a problem statement is prepared by the simulation analyst, it is important that the client understand and agree with the formulation. It is suggested that a set of assumptions be prepared by the simulation analyst and agreed to by the client. Even with all of these precautions, it is possible that the problem will need to be reformulated as the simulation study progresses.
2. **Setting of objectives and overall project plan.** Another way to state this step is prepare a proposal. This step should be accomplished regardless of location of the analyst and client, viz., as an external or internal consultant. The objectives indicate the questions that are to be answered by the simulation study. The project plan should include a statement of the various scenarios that will be investigated. The plans for the study should be indicated in terms of time, which will be required, personnel that will be used, hardware and software requirements if the client wants to run the model and conduct the analysis, stages in the investigation, output at each stage, cost of the study and billing procedures, if any.
3. **Model conceptualization.** The real-world system under investigation is abstracted by a conceptual model, a series of mathematical and logical relationships concerning the components and the structure of the system. It is recommended that modeling begin simply and that the model grow until a model of appropriate complexity has been developed. For example, consider the model of a manufacturing and material handling system. The basic model with the arrivals, queues and servers is constructed. Then, add the failures and shift schedules. Next, add the material-handling capabilities. Finally, add the special features. Constructing an unduly complex model will add to the cost of the study and the time for its completion without increasing the quality of the output. Maintaining client involvement will enhance the quality of the resulting model and increase the client's confidence in its use.

NOTES

4. **Data collection.** Shortly after the proposal is accepted a schedule of data requirements should be submitted to the client. In the best of circumstances, the client has been collecting the kind of data needed in the format required, and can submit these data to the simulation analyst in electronic format. Oftentimes, the client indicates that the required data are indeed available. However, when the data are delivered they are found to be quite different than anticipated. When the study commenced, the data delivered were the average *talk time* of the reservationist for each of the years. Individual values were needed, not summary measures. Model building and data collection are shown as contemporaneous in Fig. 2.2. This is to indicate that the simulation analyst can readily construct the model while the data collection is progressing.

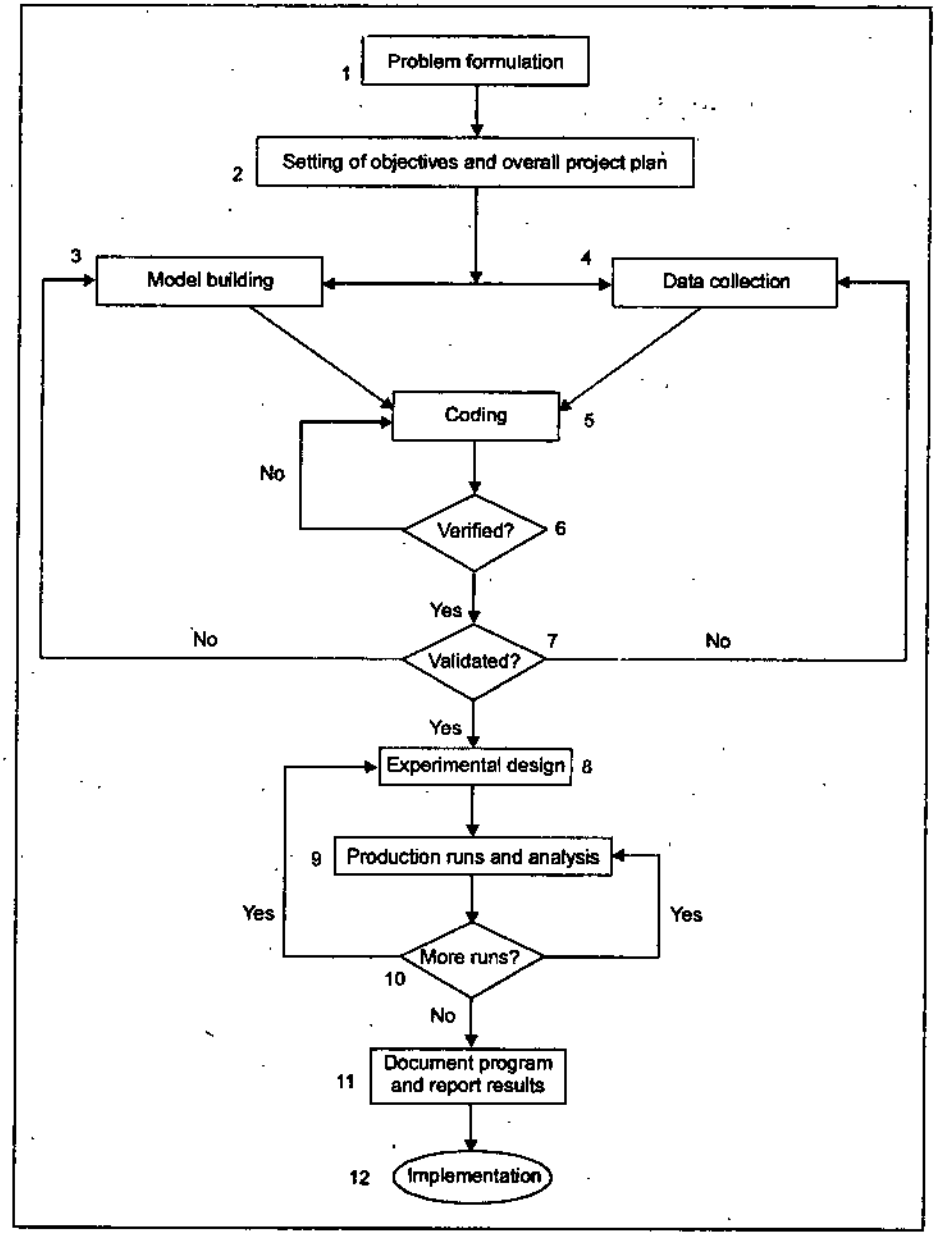

Fig. 2.2 Steps in a simulation study

NOTES

5. **Model translation.** The conceptual model constructed in Step 3 is coded into a computer recognizable form, an operational model.

6. **Verified?** Verification concerns the operational model. These models are orders of magnitude smaller than real models (say 50 lines of computer code versus 2,000 lines of computer code). It is highly advisable that verification take place as a continuing process. It is ill advised for the simulation analyst to wait until the entire model is complete to begin the verification process. Also, use of an interactive run controller, or debugger, is highly encouraged as an aid to the verification process.

7. **Validated?** Validation is the determination that the conceptual model is an accurate representation of the real system. Can the model be substituted for the real system for the purposes of experimentation? If there is an existing system, call it the base system, then an ideal way to validate the model is to compare its output to that of the base system. Unfortunately, there is not always a base system. There are many methods for performing validation.

8. **Experimental design.** For each scenario that is to be simulated, decisions need to be made concerning the length of the simulation run, the number of runs (or say replications), and the manner of initialization, as required.

9. **Production runs and analysis.** Production runs, and their subsequent analysis, are used to estimate measures of performance for the scenarios that are being simulated.

10. **More runs?** Based on the analysis of runs that have been completed, the simulation analyst determines if additional runs are needed and if any additional scenarios need to be simulated.

11. **Documentation and reporting.** Documentation is necessary for numerous reasons. If the simulation model is going to be used again by the same or different analysts, it may be necessary to understand how the simulation model operates. This will enable confidence in the simulation model so that the client can make decisions based on the analysis. Also, if the model is to be modified, this can be greatly facilitated by adequate documentation. The result of all the analysis should be reported clearly and concisely. This will enable the client to review the final formulation, the alternatives that were addressed, the criterion by which the alternative systems were compared, the results of the experiments, and analyst recommendations, if any.

12. **Implementation.** The simulation analyst acts as a reporter rather than an advocate. The report prepared in step 11 stands

NOTES

on its merits, and is just additional information that the client uses to make a decision. If the client has been involved throughout the study period, and the simulation analyst has followed all of the steps rigorously, then the likelihood of a successful implementation is increased.

2.9 TYPES OF SYSTEM SIMULATION

Continuous system and discrete system simulation are two main types of system simulation on the basis of continuous and discrete systems. We will discuss them in detail in next unit.

Real-time Simulation

The general process by which a mathematical model of an engineering system is constructed, involves understanding of physical of chemical laws. This includes a number of experiments of measurements to derive the different coefficients of the model. This can be particularly time consuming if the model is not being simplified by assuming linearity. Detailing of system experiments or measurements and preliminary work could be avoided if an actual device if an actual device can be used, rather than constructing a model.

Real-time simulation uses this approach. In Real-time simulation, actual devices are used in conjunctions with either digital or hybrid computer to provide simulation of the parts of the system that do not exist or that cannot conveniently be used in an experiment. These actual devices used in experiment should be the part of the desired system. Real-time simulation involves interaction with human being to avoid designing of human model.

Real-time simulation requires computers that can operate in real time. These computers must be able to respond immediately to signals sent from the physical devices and sent out signals at specific points in time. Real-time simulation is used in many areas such as in aerospace industry. Simulators, provide human beings with a substitute for some environment or situation. Best example of simulators is used in training astronauts. They are suspended in spring harnesses that reduce their apparent weight to that which occurs on the moon surface. Since the main purpose of such devices is to duplicate some sort of sensory stimulus, they cannot properly be classified as computers.

The Fig. 2.3, shows the variation of real time simulation from analytical solutions.

Hybrid Simulation

Hybrid Computer is combination of traditional analog-computer elements, giving smooth, continuous outputs, and elements carrying out some nonlinear, digital operations as storing values, switching, and performing

logical operations. Integration of analog computer capabilities and special-purposes or specially constructed devices is hybrid computer.

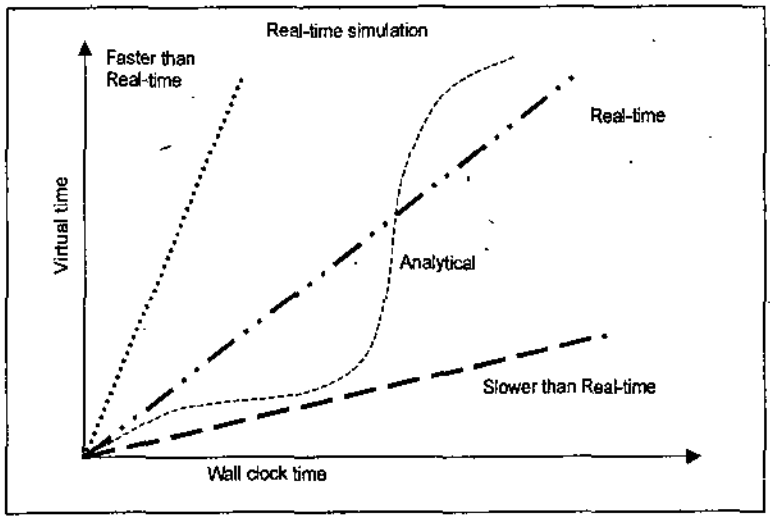

Fig. 2.3 Real-time simulations

Hybrid computers may be used to simulate systems that are mainly continuous but have some digital element also. Hybrid computers are also useful when a system that can be adequately represented by an analog computer model is the subject of a repetitive study. Hybrid computer can be arranged to carry out large portion of the study without human intervention.

For most of system studies, it is clear that model built can be either *continuous or discrete. This factor determines whether* analog or digital computers are used for system simulation. There are times when we have combined discrete-continuous simulation and so we have to combine analog and digital computers to provide hybrid simulation. The arrangement of hybrid simulation depends upon the requirements of the application.

One computer may be simulating the system being studied, whereas the other provides a simulation of the environment in which the system to operate. Simulated system is an interconnection of continuous and discrete subsystems. This system can be modeled best by linkage of an analog computer and a digital computer.

*Hybrid simulation has become easier due to the availability of mini-*computers, which has lowered the cost and allows dedicated computers for an application. High-speed converters are needed to transform signals from one from of representation to the other. Hybrid Simulation is used for the linkage of functionally distinct analog and digital computers done for the purpose of simulation.

Simulation of Stochastic Processes

In simulation of stochastic processes, we model a particular system by studying the flow of *entities* that move through that system. Entities

NOTES

can be customers, job orders, particular parts, information packets, etc. An entity can be any object that enters the system, moves through a series of processes, and then leaves the system. These entities can have individual characteristics which we will call *attributes.* An attribute is associated with the specific, individual entity. Attributes might be such things as name, priority, due date, required CPU time, ailment, account number etc. As the entity flows through the system, it will be processed by a series of *resources.* Resources are anything that the entity needs in order to be processed. For example, resources might be workers, material handling equipment, special tools, a hospital bed, access to the CPU, a machine, waiting or storage space, etc. Resources may be fixed in one location (*e.g.,* a heavy machine, bank teller, hospital bed) or moving about the system (*e.g.,* a forklift, repairman, doctor). A simulation model is therefore a computer program which represents the logic of the system as entities with attributes arrive, join queues to await the assignment of required resources, are processed by the resources, released and exit the system. In addition to the logic of how an entity flows through the system, the computer program keeps track of and advances time, as well as keeping track of resource utilization, time spent in queues, time in the system (processing time), and other desired statistics. Much of what happens in the system is probabilistic or stochastic in nature. For example the time between arrivals, the time for a resource to process the entity, the time to travel from one part of the system to another and whether a part passes inspection or not, are usually all random variables. It is these types of data for input to the model that are difficult to obtain.

To perform statistical analysis of the simulation output we need to establish some conditions, *e.g.,* output data must be a covariance stationary process (*e.g.,* the data collected over n simulation runs).

Stationary Process (strictly stationary): A stationary stochastic process is a stochastic process $\{X(t), t \in T\}$ with the property that the joint distribution all vectors of h dimension remain the same for any fixed h.

First Order Stationary: A stochastic process is a first order stationary if expected of $X(t)$ remains same for all t.

Second Order Stationary: A stochastic process is a second order stationary if it is first order stationary and covariance between $X(t)$ and $X(s)$ is function of t-s only.

Social Simulation

Social scientists have always constructed models of social phenomena. Simulation is an important method for modeling social and economic processes. There are different types of computer simulation and their

application to social scientific problems. Faster hardware and improved software have made building complex simulations easier. Computer simulation methods can be effective for the development of theories as well as for prediction. For example, macro-economic models have been used to simulate future changes in the economy; and simulations have been used in psychology to study cognitive mechanisms.

As a general approach in the field, a world is specified with much computational detail. Then the world is simulated to reveal some of the non-trivial implications of the world. When these non-trivial implications are made known in world, apparently it constitutes some added values.

Web-based Simulation

Web-based simulation is quickly emerging as an area of significant interest for both simulation researchers and simulation practitioners. This interest in web-based simulation is a natural outgrowth of the proliferation of the World Wide Web and its attendant technologies, *e.g.*, HTML, HTTP, CGI, etc. Also the surging popularity of, and reliance upon, computer simulation as a problem solving and decision support systems tools.

The appearance of the network-friendly programming language, Java, and of distributed object technologies like the Common Object Request Broker Architecture (CORBA) and the Object Linking and Embedding/ Component Object Model (OLE/COM) have had particularly acute effects on the state of simulation practice.

Currently, the researchers in the field of web-based simulation are interested in dealing with topics such as methodologies for web-based model development, collaborative model development over the Internet, Java-based modeling and simulation, distributed modeling and simulation using web technologies, and new applications.

Parallel and Distributed Simulation

The increasing size of the systems and designs requires more efficient simulation strategies to accelerate the simulation process. Parallel and distributed simulation approaches seem to be a promising approach in this direction. Major topics under such simulations are:

- Synchronization, scheduling, memory management, randomized and reactive/adaptive algorithms, partitioning and load balancing.
- Synchronization in multi-user distributed simulation, virtual reality environments, and interoperability.
- System modeling for parallel simulation, specification, re-use of models/code, and keeping parallel existing simulations.

NOTES

- Language and implementation issues, models of parallel simulation, execution environments, and libraries.
- Theoretical and empirical studies, prediction and analysis, cost models, benchmarks, and comparative studies.
- Computer architectures, telecommunication networks, VLSI, manufacturing, dynamic systems, and biological/social systems.
- Web based distributed simulation such as multimedia and real time applications, fault tolerance, implementation issues, use of Java, and CORBA.

Object-oriented Simulation

Object Oriented Simulation (OOS) can be considered as a special case of Object Oriented Programming (OOP). Some principles of OOP like existence of a varying number of instances of interfering objects have been in standard use in simulation environment for a long time, often using other terminology. The Simula language is the first true object oriented language. Being more than 38 years old, it still has most (and all important) mechanisms and principles of OOP. Some things like classes, inheritance, virtual methods, etc., have been defined in Simula long time before; they were rediscovered by the Object Oriented Programming boom in recent years.

Object-oriented simulation offers excellent tools for treating models of complex dynamic systems. First of all, we must decide when the system is really complex. Obviously, a system described by a huge set of equations is not necessarily complex. We can solve on a computer set of thousands of simultaneous differential equations, but this does not mean that the model we solve is complex. On the other hand, a system may result to be complex even if the equations are apparently simple, but the system components have very distinct dynamic behavior or if they are of different kind. We say that a dynamic system is complex, if it has multiple components that reveal different dynamic properties. This may occur, for example, when all system components are continuous with concentrated parameters, but the model includes very fast and very slow parts.

Other example is a system where discrete parts interact with continuous sub models of different speed and different kind, for example an electronic circuit that contains integrated circuits as well as electro-mechanical parts such as relays and motors. In other words, the model complexity has little to do with the model size. Using object-oriented approach we can simulate complex systems creating objects that simulate system sub models and run concurrently. The idea is to launch a set of objects and to coordinate them by other object that only connects the sub models and controls a general (global) interaction rules. Thus,

a sub model of very different kind can run and interact in the same simulation program; some of them with integration step thousands of times smaller than others and some of them being discrete or combined.

On-line Simulation

Internet together with Java and JavaScript offers incredible possibilities in problem solving. Instead of time consuming downloading and installation of software packages, it is possible to open directly various solvers, especially for problems that are not frequent and that do not require time consuming computation. Such simulations are available on Internet, for example, Silk, SLX, STARDIS, RT-LAB, JSIM, etc. In the near future it will be more and more difficult to draw a line between web simulation and traditional simulation. Web simulation must allow the simulation and interaction of distributed entities. Not all of the tools are ready for this, but they somehow aim in that direction.

2.10 SIMULATION OF PURE-PURSUIT PROBLEM

A fighter aircraft observes an enemy bomber and flies directly toward it. The main aim of fighter aircraft is to catch the bomber and destroy it. The bomber (or say the target) continues flying (along a specified curve) so that the fighter aircraft (the pursuer) has to change its direction to keep pointed towards the target. Here we determine the course of attack of the fighter and how long it would take for the catch of the bomber. If the target flies along a straight line, the problem can be solved directly with analytic techniques, but the case if the path of the target fly is curved than the problem is much more difficult and normally cannot be solved directly. Simulation is used to solve this problem. Some assumptions for simulation are following:

1. The target and the pursuer are flying in the same horizontal plane. When the fighter first sights the bomber, and both stay in that plane. This makes the pursuit model two-dimensional.
2. The fighter's speed VF is constant (say 20 kilometers/minute).
3. The target's path is specified.
4. After a fixed time span Δt, the fighter changes its direction in order to point itself toward the bomber.

We are considering a rectangular coordinate system in which the two aircrafts are flying. Distances in graph are given in kilometers and the time in minutes. We start measuring the time when the fighter first sees the bomber.

NOTES

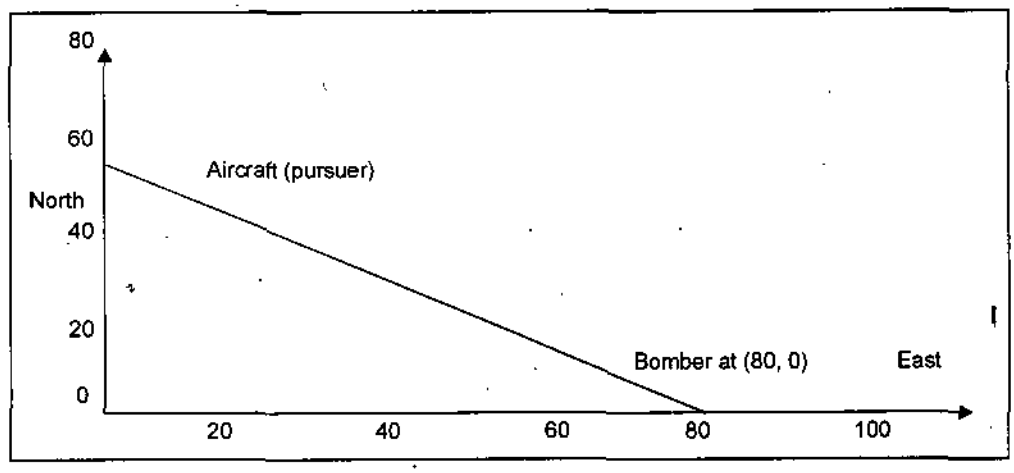

Fig. 2.4 Position of pursuer and target at time zero.

We represent the path of the bomber by two arrays XB(t) and YB(t) respectively. Likewise, we shall represent the path of the fighter by two arrays XF(t) and YF(t) respectively. Our aim is to compute the positions of the pursuer *i.e.*, XF(t) and YF(t) for t = 1, 2, 3, . . ., or until the fighter catches the bomber. Assuming that once the fighter is within 10 kilometer of the bomber, the fighter shoots down its target by firing a missile and the pursuit is over.

In case if the target is not caught up within 10 minutes, the pursuit is abandoned and the target is considered escaped. From the time t = 0 till the target is shot down, the attack course is determined as follows:

The distance $D(t)$ at a given time t between the target and the pursuer is given by

$$D(t) = \sqrt{YB(t) - YF(t)^2 + (XB(t) - XF(t)^2} \quad ...(1)$$

The angle θ of the line from the fighter to the target at a given time t is determined by

$$\sin\theta = \frac{YB(t) - YF(t)}{D(t)} \quad ...(2)$$

$$\cos\theta = \frac{XB(t) - XF(t)}{D(t)} \quad ...(3)$$

Using this value of the position of the fighter at time (t + 1) is determined by

$$XF(t + 1) = XF\ (t) + VF\ \cos\theta \quad ...(4)$$

$$YF\ (t + 1) = YF\ (t) + VF\ \sin\theta \quad ...(5)$$

With these new coordinates of the pursuer, its distance from the target is again calculated using equation (1). If this distance is 10 kilometer or less the pursuit is over, otherwise θ is recomputed, and the process continues. The simple strategy of pursuer redirecting him toward the target at fixed intervals of time, while the target

goes on its predetermined path without making any effort to evade the pursuer, is called pure pursuit. Although in many situations, the strategy used by the pursuer is more sophisticated this basic approach can be used for any pursuit problems as long as we know the path the pursuer takes and the rule by which the pursuer redirects him.

NOTES

2.11 QUEUEING SYSTEM : AN INTRODUCTION

The theory enables mathematical analysis of several related processes, including arriving at the queue, waiting in the queue (essentially a storage process), and being served by the server(s) at the front of the queue. The theory permits the derivation and calculation of several performance measures including the average waiting time in the queue or the system, the expected number waiting or receiving service and the probability of encountering the system in certain states, such as empty, full, having an available server or having to wait a certain time to be served.

The word queue comes, via French, from the Latin *cauda*, meaning tail. Most researchers in the field prefer the spelling 'queueing' over 'queuing', although the latter is somewhat more common in other contexts. Queueing Theory is the study of waiting times in queues (lines). Queueing theory is used to optimizing systems to reduce wait times. An easy example to understand is looking at wait times in line. For example, if a store has 10 different service lines then any delay at a specific line means all those in that line have to suffer to a greater extent. If, instead, there is one line that feeds all 10 stations then a delay in one line will mean that a delay at one station does not adversely impact the few unlucky enough to choose that line.

Queueing theory, the theory of congestion, is the branch of operations research which explores the relationships between demand on a service system and the delays suffered by the users of that system. Since almost all urban service systems can be viewed as queueing systems. Queueing theory plays a central role in the analysis of and planning for urban services. This chapter therefore deals with a review of some important results in queueing theory and with an introduction to the applications of these results to the problems on which this book focuses.

Queuing theory is generally considered a branch of operations research because the results are often used when making business decisions about the resources needed to provide service. It is applicable in a wide variety of situations that may be encountered in business, commerce, industry, public service and engineering. Applications are frequently encountered in customer service situations as well as transport and telecommunication and it is occasionally linked to ride theory. Queueing

theory is directly applicable to intelligent transportation systems, call centers, networks, telecommunications, server-queueing, mainframe computer queueing of telecommunications terminals, advanced telecommunications systems, and traffic flow, etc.

NOTES

Definitions and notations. Here we are giving definitions of queueing theory/model/system used world wide.

- A queue is a line of customers waiting to be served.
- We study the phenomena of standing, waiting, and serving, and we call this study Queueing Theory.
- Any system in which arrivals place demands upon a finite capacity resource may be termed a queueing system.
- The term Queueing Theory is often used to describe the more specialized mathematical theory of waiting lines.
- A Queueing System can be described as customers arriving for service, waiting for service if it is not immediate, and if having waited for service, leaving the system after being served. The primary tool for studying these problems (of congestions) is known as queueing theory.
- A queue or waiting line is formed when customers arrive at a counter offering certain facilities and demand service.
- A queue, or a waiting line, involves arriving items that wait to be served at the facility which provides the service they seek.
- In queueing theory, we study situations where units of some kind arrive at a service facility for receiving service of some description, some of the units have to wait for service, and depart after service.
- The theory of queues deals with the investigation of the stochastic law of different processes arising in connection with mass servicing in cases where random fluctuations occur.

As conclusion, basic phenomenon of queueing arises whenever a shared facility needs to be accessed for service by a large number of jobs or customers. A queue is formed when service requests arrive at a service facility and are forced to wait while the server is busy working on other requests.

In this unit we shall use the term queueing system to refer to a generic model, which comprises three elements: a user source, a queue, and a service facility that contains one or more (including possibly an infinite number of) identical servers in parallel. Thus, each user of a queueing system is "generated" by a user source, passes through the queue where he/she may remain for a nonnegative period of time (including possibly zero time), and then is processed by a single server because of the parallel arrangement of servers. Once a user has left

any one of the servers in the system, after obtaining service there, the user is considered to have left the queueing system as well.

In view of the description given above, a queueing network can now be defined as a set of interconnected queueing systems. Thus, in a queueing network, the user sources for some of the queueing systems in the network may be other queueing systems in the same network. It can also be inferred that the analysis of models of queueing systems, as we have defined them here, provides the building blocks for the analysis of queueing networks.

To describe a queueing system fully, information must be supplied about all three generic elements of the system as we see in Fig. 2.5:

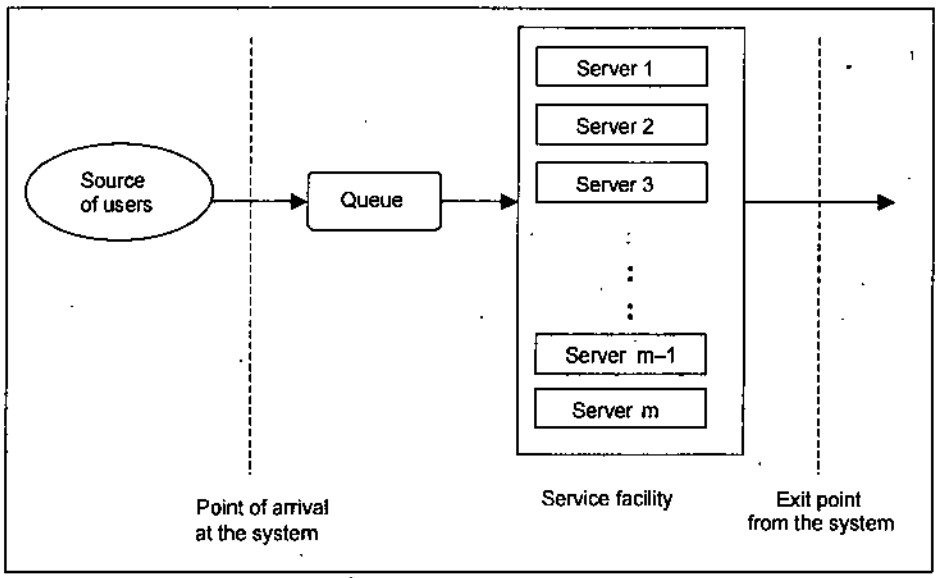

Fig. 2.5 Generic queueing system

- The user generating process (*i.e.*, the arrival process of users at the system).
- The queue discipline (*i.e.*, the order in which users obtain access to the service facility, once they join the queue).
- The service process.

To describe a queueing network, further information must be provided (see Fig. 2.6) on how the queueing systems are interconnected, how they interact, and how users are assigned to the queueing systems. It should be obvious that there exist countless variations of queueing systems and networks. We shall have occasion to refer to some of these variations in following sections. At this point, however, it should be noted that a code has been used in queueing theory for years to describe some of the simplest (and best understood) queueing systems. This code is of the form $A|B|m$, where A and B are letter symbols and m is an integer constant. The letters A and B indicate the probability

NOTES

distribution of user inter-arrival times and of service times, respectively, and m is the number of identical parallel servers in the queueing system (thus, m can take values from 1 to ∞).

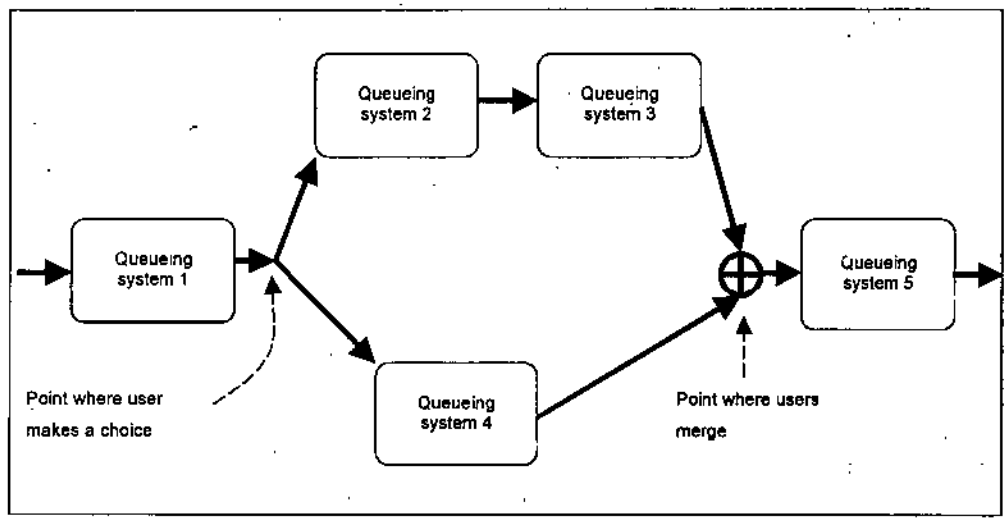

Fig. 2.6 Queueing network considering five queueing systems

The standard code letters used for probability distributions in queueing theory are:

M = Poisson (*i.e.*, negative exponential pdf for user inter-arrival times or for service times; M stands for "memory less")

D = deterministic (*i.e.*, inter-arrival or service times are constant)

E_k = kth-order Erlang distribution

H_k = kth-order hyper-exponential distribution

G = "general" distribution (*i.e.*, any distribution at all)

The letters A and B can thus be any one of the five symbols above. Examples of queueing systems that can be defined with this convention are:

- **M/M/1:** This is the simplest queueing system to analyze. Here the arrival and service time are negative exponentially distributed (Poisson process). The system consists of only one server. This queueing system can be applied to a wide variety of problems as any system with a very large number of independent customers can be approximated as a Poisson process. Using a Poisson process for service time however is not applicable in many applications and is only a crude approximation.
- **M/D/n:** Here the arrival process is Poisson and the service time distribution is deterministic. The system has n servers (for example, a ticket booking counter with n cashiers.) Here the service time can be assumed to be same for all customers).
- **G/G/n:** This is the most general queueing system where the arrival and service time processes are both arbitrary. The system

still spend the same amount of time in the restaurant (T). This will double the number of customers in the restaurant (N). By the same logic if the customer arrival rate remains the same but the customers service time doubles, this will also double the total number of customers in the restaurant.

Queueing System Classification

With Little's Theorem, we have developed some basic understanding of a queueing system. To further our understanding we will have to dig deeper into characteristics of a queueing system that impact its performance. For example, queueing requirements of a restaurant will depend upon factors like:

- How do customers arrive in the restaurant? Are customer arrivals more during lunch and dinner time (a regular restaurant)? Or is the customer traffic more uniformly distributed (*e.g.*, a cafe)?
- How much time do customers spend in the restaurant? Do customers typically leave the restaurant in a fixed amount of time? Does the customer service time vary with the type of customer?
- How many tables does the restaurant have for servicing customers?

The above three points correspond to the most important characteristics of a queueing system. They are explained below:

Table 2.2. Characteristics of a Queueing System

Arrival Process	• The probability density distribution that determines the customer arrivals in the system. • In a messaging system, this refers to the message arrival probability distribution.
Service Process	• The probability density distribution that determines the customer service times in the system. • In a messaging system, this refers to the message transmission time distribution. Since message transmission is directly proportional to the length of the message, this parameter indirectly refers to the message length distribution.
Number of Servers	• Number of servers available to service the customers. • In a messaging system, this refers to the number of links between the source and destination nodes.

Based on the above characteristics, queueing systems can be classified by the following convention:

NOTES

Facts about Queueing System

Queueing Networks. Queues can be chained to form queueing networks where the element departing from one queue enter the next queue. Queueing networks can be classified into two categories: open queueing networks and closed queueing networks. Open queueing networks have an external input and an external final destination. Closed queueing networks are completely contained and the customers circulate continually never leaving the network.

The role of the Poisson process and exponential distributions. To derive a queueing model that represents a real-life system, we prefer a form that is simple or tractable, but require that it be sufficiently realistic. For queueing theory, it has been found convenient to work with probability distributions which exhibit the memory-less property, as this commonly simplifies the mathematics involved. As a result, queueing models are frequently modeled as Poisson processes through the use of the exponential distribution.

Limitations of the mathematical approach. Classic queueing theory is often too mathematically restrictive to be able to model all real-world situations exactly. This restriction arises because the underlying assumptions of the theory do not always hold in the real world. For example; the mathematical models often assume infinite numbers of customers, or queue capacity, or no bounds on inter-arrival or service times, when it is quite apparent that these bounds must exist in reality. Often, although the bounds do exist, they can be safely ignored because the differences between the real-world and theory is not statistically significant, as the probability that such boundary situations might occur is remote compared to the expected normal situation. In other cases the theoretical solution may either prove intractable or insufficiently informative to be useful.

Alternative means of analysis have thus been devised in order to provide some insight into problems which do not fall under the mathematical scope of queueing theory, though they are often scenario-specific since they generally consist of computer simulations and/or of analysis of experimental data.

2.13 THE QUEUEING MODEL

In queueing theory, a queueing model is used to approximate a real queueing situation or system, so the queueing behavior can be analyzed mathematically. Queueing models allow a number of useful steady

follow the exponential distribution, the distribution of the service time does not. The distribution of the service time may follow any general statistical distribution, not just exponential. Relationships are still able to be derived for a number of performance measures if one knows the arrival rate and the mean and variance of the service rate. However the derivations are generally more complex. A number of special cases of M/G/1 provide specific solutions that give broad insights into the best model to choose for specific queueing situations because they permit the comparison of those solutions to the performance of an M/M/1 model.

A queueing system consists of one or more servers that provide service of some kind to arriving customers. Customers who arrive to find all servers busy (generally) join one or more queues or lines in front of the servers, hence the name queueing system. A large proportion of all discrete-event simulation studies have involved the modeling of a real-world queueing system, or a least some component of the system being simulated was a queueing system.

Components of a Queueing System

A queueing system is characterized by *three* components:

(a) Arrival process

(b) Service mechanism

(c) Queue discipline.

Specifying the arrival process for a queueing system consists of describing how customers arrive to the system. Let A_i be the inter-arrival time between the arrivals of the $(i-1)^{th}$ and i^{th} customers. If $A_1, A_2, A_3...$ are assumed to be IID random variables, we shall denote the means (or expected) *inter-arrival time* by *E(A) and call* $\lambda = \frac{1}{E(A)}$ *the arrival rate* of customers.

The *service mechanism* for a queueing system is articulated by specifying the one queue feeding all servers, and the probability distribution of customers, service times. Let S_i be the service time of the l^{th} arriving customer. If $S_1, S_2,...$ are IID random variables, we shall denote the mean service time of a customer by E(S) and call ω = 1/E(S) the *service rate* of a server:

(a) FIFO : Customers are served in a first-in, first-out manner.

(b) LIFO : Customers are served in last-in, first-out manner.

(c) Priority : Customers are served in order of their importance or on the basis of their service requirements.

Consider a single-server queueing system as shown in Fig. 2.7 for which the inter-arrival times $A_1, A_2, A_3...$are *independent and identically distributed* (IID) random variables. A customer who arrives and finds

NOTES

the server idle enters service immediately, and the service times $S_1, S_2, \ldots$, of the successive customers are IID random variables that are independent of the inter-arrival times. A customer who arrives and finds the server busy joins the end of a single queue. Upon completing service for a customer, the server chooses a customer from the queue (if any) in a first-in, first-out (FIFO) manner.

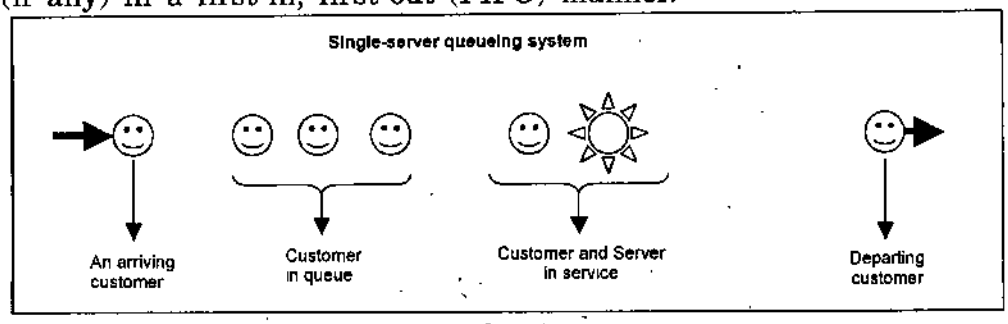

Fig. 2.7 Single-server queueing system

Arrivals and services are described by the distribution of the time between arrivals and service times. The overall effective arrival rate must be less than the maximum service rate, or the waiting line will grow without bound. When queues grow without bound, they are termed "explosive" or unstable. An exceptional situation would be arrival rates that are greater than service rates for short period of time. The *state of the system* is the number of units in the system and the status of the server, busy or idle. An *event* is set of circumstances that cause in instantaneous change in the state of the system. In the single server queueing system there are *two* possible events that can affect the state of the system. They are:

- Entry of unit into the system *i.e.,* the arrival event.
- The completion of service on a unit *i.e.,* the departure event.

The queueing system includes:

- The server
- The unit being serviced
- Units in the queue (if any units are waiting).

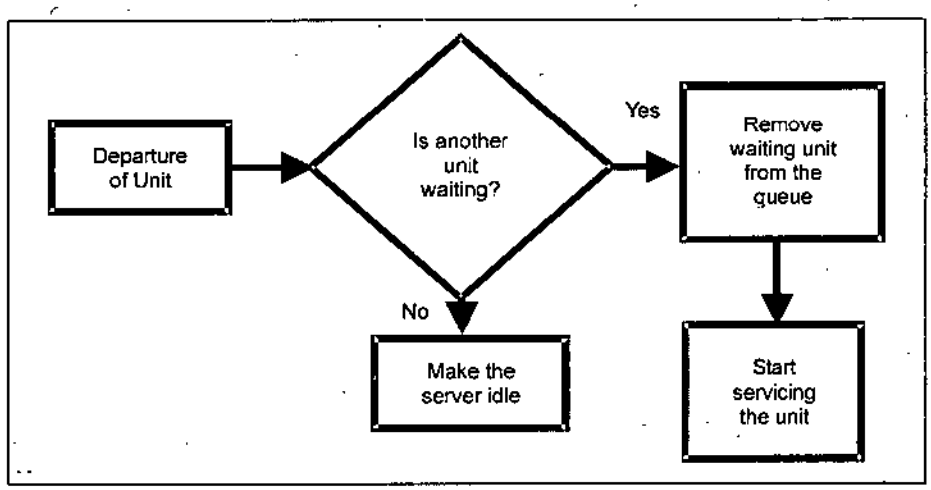

Fig. 2.8 Flow diagram of service just completed

The server can have only *two* possible conditions:

- Busy
- Idle

If service has been completed, the simulation proceeds as shown in Fig. 2.9.

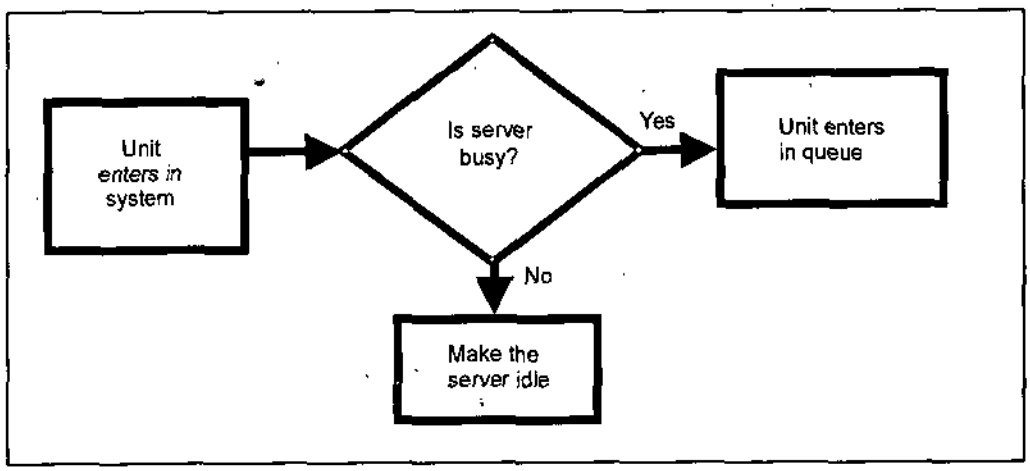

Fig. 2.9 Flow diagram of unit entering system

The next event that occurs after service just completed is new unit entering the system. The entered unit may find the server either idle or busy. If the server is busy, the unit enters the queue. If the server is idle and the queue is empty, the unit enters the server. It is impossible for the server to be idle and the queue to be not empty. The figure given above shows the course of action taken when a unit enters the system. From this flow diagram a Table 2.3 (below) can be drawn with columns as queue status and rows as server status.

Table 2.3. Action taken on Unit Arrival Event

Queue status / Server Status	*Non Empty*	*Empty*
Server Busy	Enter into Queue	Enter into Queue
Server Idle	Impossible	Enter service

After the completion of a service, the server may become idle or remain busy with the next unit. The course of action as described in points below:

(*a*) If the queue is not empty, another unit will enter the server and it will be busy.

(*b*) If the queue is empty, when all the services are complete including the servicing unit then the server will be idle.

NOTES

(c) It is impossible for the server to become busy, if the queue is empty, when all the services are complete.

(d) It is impossible for the server to be idle after a service is completed, when the queue is not empty.

Above stated point can be shown in tabular form as in Table. 2.4

Table 2.4. Server Outcomes after Unit Service Completion

Queue status / *Server Outcomes*	*Non Empty*	*Empty*
Server Busy	Enter into Queue	Impossible
Server Idle	Impossible	No unit

Simulations of queueing system generally require the maintenance of an event list for determining what happens next. The event list indicates the times at which the different types of event occur for each unit in the queueing system. The times are "clock" that marks the occurrence of events in time. In simulation, the events usually occur at random. For example, it is not known how long the bank teller will take to complete a transaction or when the next customer will arrive at a grocery checkout counter.

Random numbers are distributed uniformly and independently on the interval [0, 1]. Random digits can be used to form random number by selecting the proper number of digits for each random number and placing decimal point to the left of the value selected. Random numbers can also be generated. When numbers are generated using a set procedure they are often called as *pseudo-random numbers*. As the method is known, it is always possible to know what the sequence of numbers will be prior to the simulation. In a single-server queueing system inter-arrival times and service times are determined from the distributions of these random variables.

Let us simulate the arrival and servicing of N customer by a single server. Let the inter-arrival time of i^{th} customer be AT_i, denotes the time gap between the arrivals of the $(i - 1)^{th}$ customer and the i^{th} customer into the system.

Let the service time of i^{th} customer is ST_i where $i = 1, 2, ..., N$.

Let cumulative arrival time of i^{th} customer is CAT_i.

Service times (ST_i) and inter-arrival times (AT_i) are generated as samples from some specified probability distribution.

Assuming, initially there is no queue and the server is free. When the first customer arrives, at time zero, directly goes to the service counter. After ST_1 time, the first customer departs from the service counter.

Second customer arrives at $CAT_2 = AT_2$ ...(6)

If $ST_1 > CAT_2$ *i.e.,* service time of first customer is greater than arrival time of second customer than the second customer has to wait in the queue.

NOTES

Let WT_i is the waiting time of i^{th} customer

$$WT_2 = ST_1 - CAT_2 \quad ...(7)$$

If $ST_1 < CAT_2$ *i.e.,* service time of first customer is less than arrival time of second customer. This means, departure of first customer takes place before the arrival of second customer. Thus the service counter is free waiting for the second customer to arrive. The idle time is given by

$$IDT_2 = CAT_2 - ST_1 \quad ...(8)$$

Let us consider that $(j-1)$ customers have arrived and $(k-1)$ customers have left the system. The next customer to arrive is j^{th} and the next customer to depart is k^{th}.

$$1 \le j \le k \le N \quad ...(9)$$

$\therefore$ The queue length is given by

$$k - j - 1 \quad \text{if } k > j \quad ...(10)$$

The next arrival time is

$$NAT = CAT_j \quad ...(11)$$

The next departure time is

$NDT = CDT_k$ = Cumulative arrival time of k + waiting time of k = service time of k.

$$NDT = CDT_k = CAT_k + WT_k + ST_k \quad ...(12)$$

Whether j would arrive first of k would depart first depends on difference between them.

$$DIF = NAT - NDT \quad ...(13)$$

If difference DIF is positive than

(*a*) queue length is $k - j - 1$

(*b*) departure would take place first.

(*c*) Reduce the length of queue by 1 as departure takes place first.

(*d*) If the length of queue is zero than the service counter will remain free waiting for a customer to arrive.

If difference DIF is negative than

(*a*) arrival takes place first.

(*b*) length of the queue is increased by one.

If difference DIF is zero than

(*a*) the next arrival and departure takes place simultaneously.

(*b*) the length of the queue remains same.

NOTES

Some of the finding for short-term simulations are as follows :

(i) The average waiting time (AWT) of the customer is calculated as follows :

Average Waiting Time (in minutes) =

$$\frac{\text{Total time customer wait in queue (in minutes)}}{\text{Total number of customers}} \quad ...(14)$$

(ii) The probability that a customer has to wait in a queue is calculated as :

$$\text{Probability (wait)} = \frac{\text{Number of customers who wait}}{\text{Total number of customers}} \quad ...(15)$$

(iii) The probability that a server is idle can be calculated as :

$$\text{Probability of idle Server} = \frac{\text{Total run time of server (minutes)}}{\text{Total run time of simulation (minutes)}} \quad ...(16)$$

(iv) The probability of server being busy is calculated as :

Probability of Busy Server = 1 – Probability of idle server

Probability of Busy Server =

$$1 - \frac{\text{Total run time of server (minutes)}}{\text{Total run time of simulation (minutes) (AST)}} \quad ...(17)$$

(v) The average service time (AST) is calculated as :

$$\text{Average service time (minutes)} = \frac{\text{Total service time (minutes)}}{\text{Total number of customers}} \quad ...(18)$$

(vi) Average time between arrival (minutes) =

$$\frac{\text{Sum of all time betwen arrivals (minutes)}}{\text{Number of arrivals} - 1} \quad ...(19)$$

One is subtracted from the denominator because the first arrival is assumed to occur at time 0.

(vii) The average waiting time of those who wait is calculated as :

Average waiting time of those who wait (minutes) =

$$\frac{\text{Total time customers wait in queue (minutes)}}{\text{Total number of customers}} \quad ...(20)$$

(viii) The average time a customer spends in the system is calculated in two ways.

(a) Average time customer spends in the system =

$$\frac{\text{Total time customers spend in the system (minutes)}}{\text{Total number of customers}} \quad ...(21)$$

(*b*) Average time customer spends in the system (minutes) = Average time customer spends waiting in the queue (minutes) + Average time customer spends in service (minutes) ...(22)

Example. Single-Server Queue

A small grocery store has only one checkout counter. Assume that the time between customer arrivals is recorded as in Table 2.5. This table has five inter-arrival times are used to compute the arrival times of sing customers at the queueing system.

Table 2.5. Inter-arrival and Clock Times

Customer	*Inter-arrival time*	*Arrival time on clock*
1	–	0
2	2	2
3	4	6
4	1	7
5	2	9
6	6	15

Table 2.6 contains service times generated at random from a distribution of serving times. Only one, two, three, and four units are possible service times.

Table 2.6. Service Time

Customer	*Service time*
1	2
2	1
3	3
4	2
5	1
6	4

Solution:

From Tables 2.5. The first customer is assumed to arrive at clock time 0. This starts the clock operation. The second customer arrives two time units later, at a clock time of 2. The third customer arrive four time units later at a clock time of 6; and so on.

From Table 2.6. The first customer arrives at clock time 0, and immediately begins service, which requires two minutes. Service is completed at clock time 2. The second customer arrives at clock time 2 and is finished at clock time 3. The fourth customer arrived at clock

NOTES

time 7, but service could not begin until clock time 9. This happens as customer 3 did not finish service until clock time 9.

Table 2.7 is specifically designed for a single-server queue which server customers on a first in, first out (FIFO) basis. It keeps track of clock time at which each event occurs. The second column records the clock time of each arrival event, while the last column records the clock time of each departure event.

Fig. 2.10 shows the number of customers in the system at the various number of clock times. Table 2.8 is ordered by clock time, in which cash the events may or may not be ordered by customer number.

Table 2.7. Simulation Table

Customer (1)	*Arrival time (clock)* (2)	*Time service begin (clock)* (3)	*Service time -(duration)* (4)	*Time service ends (clock)* (5)
1	0	0	2	2
2	2	2	1	3
3	6	6	3	9
4	7	9	2	11
5	9	11	1	12
6	15	15	4	19

Column (5) is calculated as follows:

Column 5 = Column 3 + Column 4.

Column (3) is calculated verifying that if the service time is greater than the arrival time than the customer enters the queue and the time service begins is after the completion of the previous customers. Event type can be arrival and departure of the customer regarding the clock time:

1. Customer 1 is in the system from clock time 0 to clock time 2.
2. Customer 2 arrives at clock time 2 and departures at clock time 3.
3. There is no customer in the system from clock time 3 to clock time 6. Customer 3 arrivers at clock time 6 and at clock time 7 customer 4 arrives and so from 7 to 9, there are two customers (customer 3 and customer 4) in the system.
4. At clock time 9, two events are occurring simultaneously *i.e.*, departure of customer 3 and arrival of customer 5. Similarly at clock time 2, customer 1 departs and customer 2 arrives.
5. Till clock period 11, there are two customers *i.e.*, customer 4 and customer 5.

NOTES

6. At clock period 11, customer 4 departs and at clock period 12 customer 5 departs. From clock period 12 to clock period 15 there is no customer in the system. Customer 6 arrives at clock period 15 and departs at clock period 19 i.e., only one customer in the system during this clock duration.

Table 2.8. Chronological Order of Events in Single-server Queueing System

Event type number	*Clock time*
Arrival 1	0
Departure1	2
Arrival 2	2
Departure 2	3
Arrival 3	6
Departure 4	7
Arrival 3	9
Departure 5	9
Arrival 4	11
Departure 5	12
Arrival 6	15
Departure 6	19

Table 2.9. Simulation Table for Queueing Problem

Customer	*Time since last arrival (in minutes)*	*Arrival time (in minutes)*	*Service time (in minutes)*	*Time of service begin*	*Time customer wait in queue (in minutes)*	*Time service ends*	*Time customer spends in system (in minutes)*	*Idle time of service*
1	–	0	2	0	0–0=0	0+2=2	2–0=2	0–0=0
2	2–0=2	2	1	2	2–2=0	2+1=3	3–2=1	2–2=0
3	6–2=4	6	3	6	6–6=0	6+3=9	9–6=3	6–3=3
4	7–6=1	7	2	9	9–7=2	9+2=11	(11–9)+2=4	9–9=0
5	9–7=2	9	1	11	11–9=2	11+2=13	(12–11)+ 2=3	11–11=0
6	15–9=6	15	4	15	15–15=0	15+4=19	(19–15)+ 0=4	15–12=3
Total			13		4		17	

NOTES

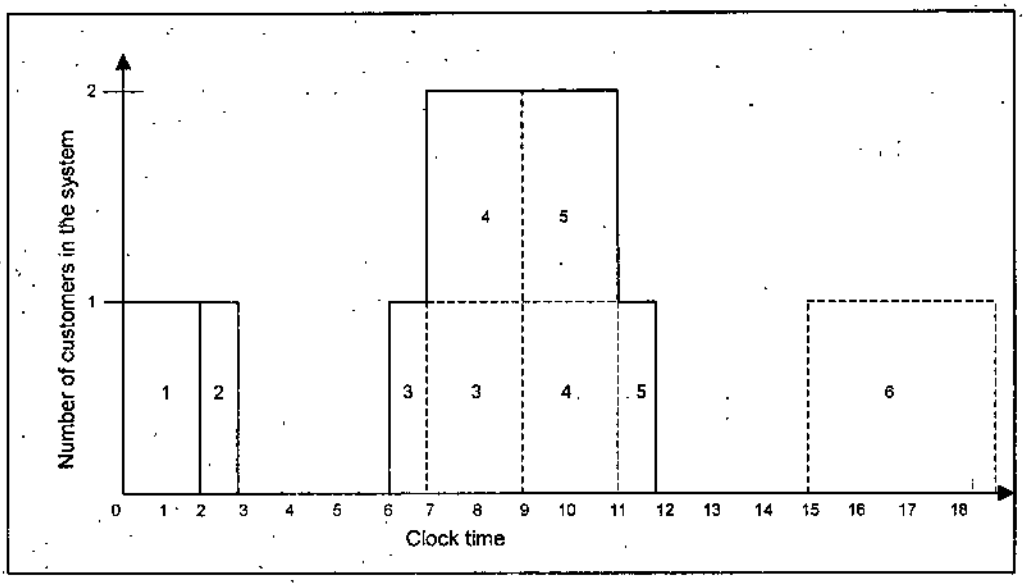

Fig. 2.10 Number of customer in the system at various clock time.

(*i*) Let us now calculate the parameters for short time simulation for this example

Average waiting time (minutes) =

$$\frac{\text{Total time customer wait in queue (minute)}}{\text{Total number of customers}}$$

Average waiting time (minutes) = $\frac{4}{6}$.

Average waiting time = 0.667 minutes.

(*ii*) Probability that a customer has to wait in the queue is

$$\text{Probability (wait)} = \frac{\text{Number of customers who wait}}{\text{Total number of customers}}$$

$$\text{Probability (wait)} = \frac{2}{6} = 0.33$$

There are only two customers *i.e.*, customer 4 and customer 5 who have to wait for their service to begin.

(*iii*) Probability of idle server

$$= \frac{\text{Total run time of server (minutes)}}{\text{Total run time of simulation (minutes)}}$$

$$\text{Probability of idle server} = \frac{6}{19}$$

Probability of Idle Server = 0.316.

(*iv*) Probability of Busy Server = 1 – Probability of Idle Server.

= 1 – 0.316

Probability of Busy Server = 0.684.

(*v*) Average service time (minutes) $= \frac{\text{Total service time (minutes)}}{\text{Total number of customers}}$

$$= \frac{13}{6}$$

Average service time (minutes) = 2.167

(*vi*) Average time between arrivals (minutes)

$$= \frac{\text{Sum of all times between arrival (min.)}}{\text{number of arrivals} - 1}$$

NOTES

$$\text{Average time between arrivals (minutes)} = \frac{15}{(6-1)}$$

Average time between arrivals = 3 minutes.

(*vii*) Average waiting time of those who wait (minutes)

$$\text{Total time of a customer} = \frac{\text{wait in queue (minutes)}}{\text{Total number of customes}}$$

$$= \frac{4}{2}\text{(minutes)}$$

Average time waiting time of those who wait = 2 minutes.

(*a*) Average time customer spends in the system (minutes)

$$\text{Total time of customer} = \frac{\text{spend in the system (minutes)}}{\text{Total number of customers}}$$

$$= \frac{17}{6}\text{minutes}$$

Average time customer spends in the system = 2.833 minutes.

(*b*) Average time customer spends in the system (minutes) = Average time customer spends waiting in the queue + Average time customer spends in service.

Substituting values from (*i*) and (*v*) and *vii* (*b*)

= 0.667 minutes + 2.167 minutes.

A decision maker would be interested in results of this type, but a longer simulation would increase the accuracy of the findings.

Some subjective inferences can be drawn from the above calculations:

(a) Less customer has to wait.

(b) Average waiting time is not excessive.

(c) Server remains idle for almost one-third of the system cycle.

(d) Probability of waiting of a customer so that server is free is also less.

(e) Cost estimation of waiting in the system can be done.

The Event Scheduling World View

In the *Event Scheduling* world view, a model describes, for each of the events, the event's effect:

- on the state,
- on the future behavior of the system. This is achieved by *scheduling* new events into the future.

NOTES

An event scheduling model for the single queue, single server example is given below:

```
declare variables:
t : Time
queue_length : PosInt
cashier_state : {Idle, Busy}
declare events:
start, arrival, departure, end
define events:
start event:
/* scheduled first automatically by simulator */
/* initializations */
queue_length = 0
cashier_state = Idle
/* schedule end of simulation */
schedule end absolute end_time
/* schedule first arrival */
schedule arrival relative 0
arrival event:
schedule arrival relative Random(IATmean, IATspread)
if (queue_length == 0)
if (cashier_state == Idle)
cashier_state = Busy
schedule departure relative Random(SERVmean, SERVspread)
else
queue_length++
else /* for queue_length != 0 */
queue_length++
departure event:
if (queue_length == 0)
cashier_state = Idle
else /* for queue_length != 0 */
queue_length--
schedule departure relative Random(SERVmean, SERVspread)
end event:
/* terminates simulation */
/* process/output performance metrics */
```

print time, queue_length /* current */

print average_queue_length

The Activity Scanning World View

NOTES

In the *Activity Scanning* world view, a model describes *conditions* which will activate *activities*. This representation (and its semantics described below) resembles that used in declarative AI (Artificial Intelligence) languages such as Prolog. An activity scanning model for the single queue, single server example is given below.

/*declare (and initialize) variables: */

t : Time

queue_length : PosInt = 0

cashier_state : {Idle, Busy} = Idle

t_arrival : Time = 0

t_depart : Time = plusInf

declare activities:

queue_pay, depart, end

queue_pay activity

condition: t >= t_arrival

actions:

if (queue_length == 0)

if (cashier_state == Idle)

keep queue_length == 0

cashier_state = Busy

t_depart = t + Random(SERVmean, SERVspread) /* service time */

else

queue_length++

else /* queue_length != 0 */

queue_length++, keep cashier_state == Busy

t_arrival = t + Random(IATmean, IATspread) /* inter arrival time */

depart activity

condition: t >= t_departure

actions:

if (queue_length == 0)

cashier_state = Idle

NOTES

```
else /* queue_length != 0 */
queue_length—, keep cashier_state == Busy
t_depart = t + Random(SERVmean, SERVspread) /* service time */
end activity
condition: t >= t_end
actions:
print t, queue_length /* current */
print avg_queue_length /* performance metric */
```

2.14 INVENTORY SYSTEM : AN INTRODUCTION

Inventory Control is designed to support the requisition processing, inventory management, purchasing, and physical inventory reconciliation functions of inventory management through a set of highly interactive capabilities. The design of Inventory Control is based on the following key objectives:

- To provide information on the availability of stocked items and the status of stocked requisitions.
- To facilitate timely requisition processing.
- To automatically record and service backorders.
- To help minimize inventory investments consistent with service objectives by purchasing decisions on usage history.
- To provide automated tools to assist servicing, purchasing, and management of the inventory.
- To improve financial control of the inventory by charge backs to the user organizations.
- To improve financial control of the inventory by periodic reconciliation of the inventory balances with the physical counts.

Inventory Control utilizes a set of user-maintained master tables, a set of system-maintained master tables, transaction document types, and offline programs to meet these objectives. It also creates inventory control management reports.

Inventory management refers to short term decisions regarding supplies, inventories, production levels, and operations infrastructure. It is an important activity towards ensuring smooth production process in organizations. An efficient inventory management system facilitates in optimum utilization of resources.

Most small businesses underestimate the importance of their inventory or of inventory management software. They do not realize that many

NOTES

of the headaches are caused by a lack of control and knowledge of their inventory. Whether it is a lack of knowledge of the quantity of a certain product, businesses continue to use outdated systems that do not allow them to get the most out of their inventory. Inventory management systems when used properly; allow a business to make established analysis of products and markets that help them make better business decisions. That is why we need to simulate such systems. They keep a tight customer control to allow business owners to better serve their customers, and keep a detailed and accurate record of their customers purchase history as well as a way to properly reorder inventory. With a tight control of inventory, owners know when products are been stolen, when products are not rotating properly, or when products need to be reordered. By default, the business begins to develop an organization and by a control of inventory, profits begin to increase. Inventory Management Systems allow programmed owners to make decisions about promotions and specials and let owners know when a product is no longer being profitable.

2.15 INVENTORY SYSTEM

In an inventory control method and system, changes of sales for individual goods are forecasted and the excess or deficiency of a stock of each of the goods at the present point of time is estimated from the results of forecast. In order to facilitate an inventory control, merchandise information is sorted and displayed in accordance with the degree of urgency, the degree of importance or the like of inventory adjustment on the basis of the results of estimation.

In inventory system what is claimed is:

1. An inventory control method comprising computer aided steps of:
   - forecasting a cumulative total of sales for goods by using a result of sales from other goods which are selected and inputted with the use of an interactive graphics system, which may exhibit a similar sales change pattern to the goods;
   - obtaining a sum of an actual result of sales and an actual stock for the goods;
   - comparing the sum with the forecasted cumulative total of sales for the goods by displaying the forecasted information on a graph as reference information and comparing the reference information to actual inventory information;

NOTES

- determining an excess or deficiency of the actual stock for the goods on the basis of the graphic comparing;
- outputting information of the goods corresponding to the determined excess or deficiency of the actual stock inventory for the goods, wherein said outputted information is in the form of a percent ratio and is labeled, the stock warning index, to provide the necessary level of priority and degree of urgency corresponding to the goods; and,
- adjusting the actual stock inventory in accordance with the outputted information.

2. The inventory control method according to claim 1, wherein the result of sales from the other goods is used as a sales change model pattern, and further including creating a whole model pattern by using an interactive graphics system.
3. An inventory control system comprising:
   - a means for forecasting a cumulative total of sales for goods by using a result of sales stored in a database for other goods which may exhibit a similar sales change pattern to the goods;
   - a means for obtaining a sum of an actual result of sales and an actual stock for the goods stored in a sales/stock results file;
   - a means for interactively and graphically comparing the obtained sum with the forecasted cumulative total of sales for the goods;
   - a means for determining an excess or deficiency of the actual stock inventory for the goods on the basis of a result from the comparing means;
   - a means for outputting information comprising a stock warning index, which prioritizes an urgency of adjusting the actual stock inventory resulting from the determined excess or deficiency of the actual stock inventory for the goods; and,
   - a means for adjusting the actual stock inventory in accordance with the stock warning index.
4. The inventory control system according to claim 3, wherein in the means for forecasting the result of sales for the other goods stored in the sales/stock results file, is used as a sales change model pattern.
5. An inventory control system comprising:
   - a means for forecasting a change in sales for goods that are subject to inventory control, by using a model pattern of a predetermined change of sales for the goods stored in a time-to-sales-correspondence file;

- a means for monitoring a difference between the forecasted change in sales for the goods and an actual change in sales for the goods stored in a sales/stock results including means for daily monitoring by calculation of a stock warning index;
- a means for outputting the stock warning index for allowing modification of the model pattern if the monitored difference is larger than a predetermined level; and,
- a means for adjusting actual stock inventory in accordance with said outputted stock warning index.

6. The inventory control system according to claim 5, wherein the forecasting means is operated before the goods are on sale.

2.16 BASIC FUNCTION OF INVENTORY

The basic function of stock (inventory) is to insulate the production process from changes in the environment as shown in Fig. 2.11. One point to note from this diagram is that most of the activities are a cost — it is only at the final point (sales of finished goods) that we get revenue to set against our costs and hopefully make a profit (= revenue-cost). Hence if we have cost associated with stock we need to deal with that stock in an *Effective*, *Efficient* and *Economic* manner.

We shall consider the problem of ordering raw material stock but the same basic theory can be applied to the problem of deciding the finished goods stock; and deciding the size of a batch in a batch production process.

The costs that we need to consider so that we can decide the amount of stock to have can be divided into stock holding costs and stock ordering (and receiving) costs as below. here conventionally, management costs are ignored.

Holding costs—associated with keeping stock over time

- storage costs
- rent/depreciation
- labor
- overheads (*e.g.*, heating, lighting, security)
- money tied up (loss of interest, opportunity cost)
- obsolescence costs (if left with stock at end of product life)
- stock deterioration (lose money if product deteriorates whilst held)
- theft/insurance

Ordering costs—associated with ordering and receiving an order

NOTES

- clerical/labor costs of processing orders
- inspection and return of poor quality products
- transport costs
- handling costs

Note here that a stock out occurs when we have insufficient stock to supply customers. Usually stock outs occur in the order lead time, the time between placing an order and the arrival of that order. Given a stock out the order may be lost completely or the customer may choose to backorder, *i.e.*, to be prepared to wait until we have sufficient stock to supply their order.

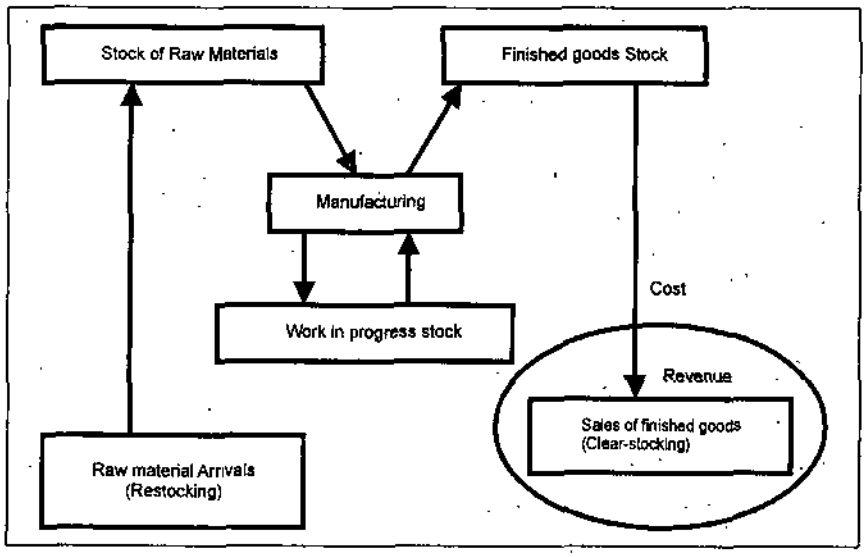

Fig. 2.11 Basic function of inventory

In this basic model we have the situation where company orders from an outside supplier that outside supplier delivers to us precisely the quantity we ask for; and we pass that stock onto customers either external customers, or an internal customer within the same company (*e.g.*, if ordering raw materials for use in the production process).

Let us assume:

- R = Stock used up at a constant rate (units per year)
- c_o = Fixed set-up cost for each order—often called the order cost
- c_h = Variable stock holding cost per unit per year.

let us assume that no lead time between placing an order and arrival of the order, and the amount to order each time, often called the *batch* (or *lot*) size. With these assumptions the graph of stock level over time takes the form shown below.

Consider drawing a horizontal line at $Q/2$ in the above diagram. If we were to draw this line then it is clear that the times when stock

exceeds $Q/2$ are exactly balanced by the times when stock falls below $Q/2$. In other words we could equivalently regard the above diagram as representing a *constant* stock level of $Q/2$ over time. Hence we have:

Annual holding cost = $c_h(Q/2)$ ...(23)

where $Q/2$ is the average (constant) inventory level.

Annual order cost = $c_o(R/Q)$ ...(24)

where (R/Q) is the number of orders per year (R used, Q each order)

So total annual cost = $c_h(Q/2) + c_o(R/Q)$ ...(25)

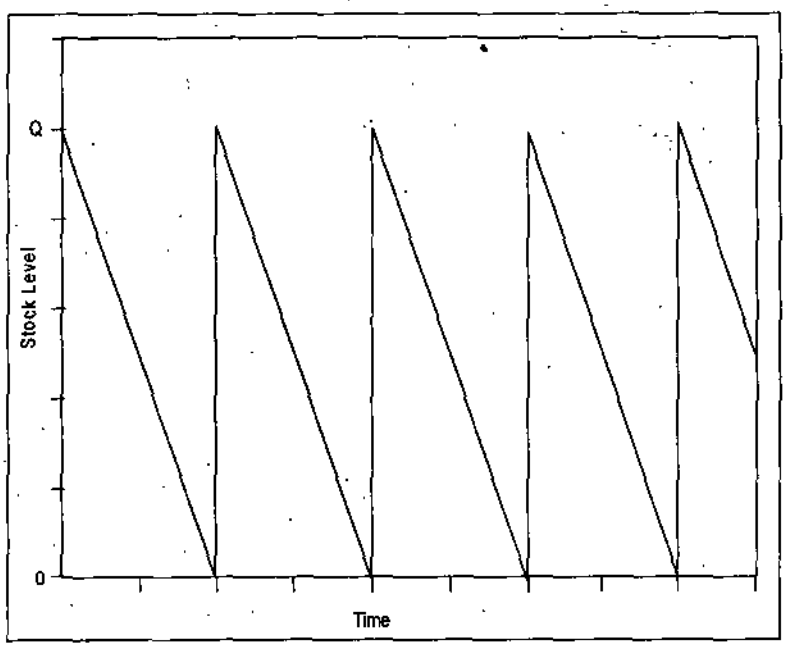

Fig. 2.12 Stock level over time

Total annual cost is the function that we want to minimize by choosing an appropriate value of Q. Obviously, there is a purchase cost associated with the R units per year. However this is just a constant as R is fixed so we can ignore it here. The diagram below illustrates how these two components (annual holding cost and annual order cost) change as Q, the quantity ordered, changes. As Q increases holding cost increases but order cost decreases. Hence the total annual cost curve is as shown below somewhere on that curve lies a value of Q that corresponds to the minimum total cost.

We can calculate exactly which value of Q corresponds to the minimum total cost by differentiating total cost with respect to Q and equating to zero.

$\partial(\text{total cost})/\partial Q = c_h/2 - c_o R/Q^2 = 0$ for minimization ...(26)

which gives

$$Q^2 = 2c_o R/c_h \quad ...(27)$$

NOTES

Hence the best value of Q (the amount to order = amount stocked) is given by $Q = (2Rc_o/c_h)^{0.5}$ and this is known as the *Economic Order Quantity (EOQ)*

To get the total annual cost associated with the EOQ we have from before that total annual cost = $c_h(Q/2) + c_o(R/Q)$, so putting $Q = (2Rc_o/c_h)^{0.5}$ into this we get that the total annual cost is given by

$$c_h((2Rc_o/c_h)^{0.5}/2) + c_o(R/(2Rc_o/c_h)^{0.5}) = (Rc_oc_h/2)^{0.5} + (Rc_oc_h/2)^{0.5} = (2Rc_oc_h)^{0.5} \quad ...(28)$$

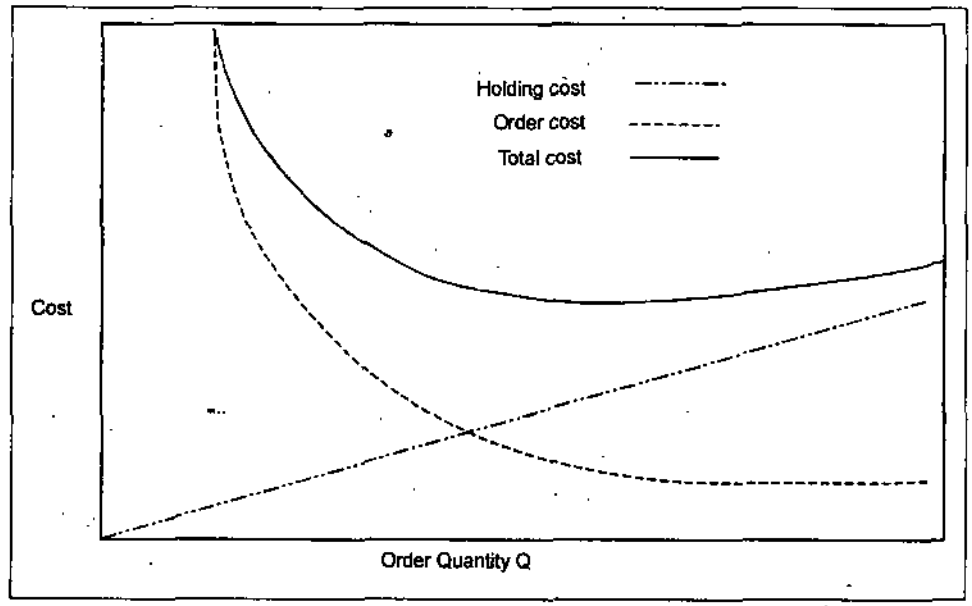

Fig. 2.13 Holding cost, order cost, and total cost in an inventory system

Hence total annual cost is $(2Rc_oc_h)^{0.5}$ which means that when ordering the optimal (EOQ) quantity we have that total cost is proportional to the square root of any of the factors (R, c_o and c_h) involved. For example, if we were to reduce c_o by a factor of 4 we would reduce total cost by a factor of 2 (note the EOQ would change as well). This, in fact, is the basis of Just-in-Time (JIT), to reduce (continuously) c_o and c_h so as to drive down total cost.

Example

A retailer expects to sell about 200 units of a product per year. The storage space taken up in his premises by one unit of this product is costed at \$20 per year. If the cost associated with ordering is \$35 per order what is the economic order quantity given that interest rates are expected to remain close to 10% per year and the total cost of one unit is \$100.

We use the EOQ formula,

$$EOQ = (2Rc_o/c_h)^{0.5} \quad ...(29)$$

Here, $R = 200$,

$c_o = 35$

and the holding cost c_h is given by

c_h = \$20 (direct storage cost per unit per year)
+ \$100 × 0.10

This term the money interest lost if one unit sits in stock for one year, *i.e.*,

c_h = \$30 per unit per year

Hence,

$$EOQ = (2Rc_o/c_h)^{0.5} = (2 \times 200 \times 35/30)^{0.5} = 21.602$$

But as we must order a whole number of units we have that:

$$EOQ = 22$$

We can illustrate this calculation by reference to the diagram below which shows order cost, holding cost and total cost.

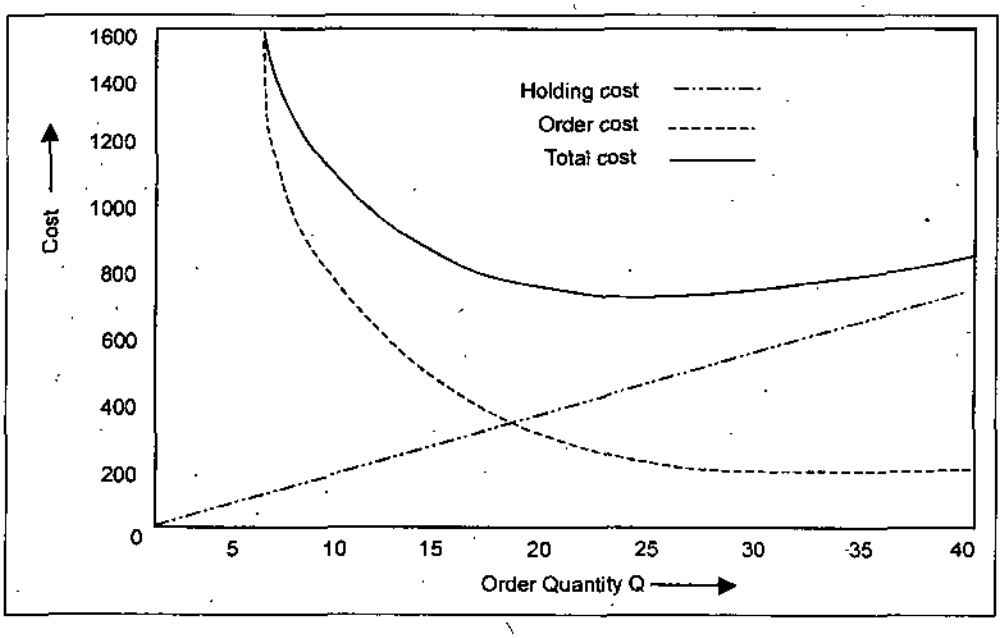

Fig. 2.14 Reference diagram showing order cost, holding cost and total cost

With this EOQ we can calculate our total annual cost from the equation

$$\text{Total annual cost} = c_h(Q/2) + c_o(R/Q) \quad ...(30)$$

Hence for this example we have that

Total annual cost = (30 × 22/2) + (35 × 200/22) = 330 + 318.2 = \$648.2

If we had used the exact Q value given by the EOQ formula (*i.e.*, Q = 21.602) we would have had that the two terms relating to annual holding cost and annual order cost would have been exactly equal to each other *i.e.*,

holding cost = order cost at EOQ point

or $(c_hQ/2) = (c_oR/Q)$ so that $Q = (2Rc_o/c_h)^{0.5}$ ...(31)

In other words, as in fact might seem natural from the shape of the Holding Cost and Order Cost curves, the optimal order quantity coincides with the order quantity that exactly balances Holding Cost and Ordering Cost. Note however that this result only applies to certain simple

NOTES

situations. It is not true (in general) that the best order quantity corresponds to the quantity where holding cost and ordering cost are in balance.

Example

Suppose, for administrative convenience, we ordered 20 and not 22 at each order—what would be cost penalty for deviating from the EOQ value?

With a Q of 20 we look at the total annual cost = $(c_h Q/2) + (c_o R/Q)$

$= (30 \times 20)/2 + (35 \times 200/20)$

$= 300 + 350 = \$650$

Hence the cost penalty for deviating from the EOQ derived value of 22 and ordering 20 at each order is \$650 – \$648.2 = \$1.8

Extensions

In order to illustrate extensions to the basic EOQ calculation we will consider the following example:

A company uses 12,000 components a year at a cost of 5 cents each. Order costs have been estimated to be \$5 per order and inventory holding cost is estimated at 20% of the cost of a component per year.

What is the EOQ?

Here R = 12000, c_o = 5 and as the inventory holding cost is 20% per year the annual holding cost per unit

c_h = cost per unit × 20%

= \$0.05 × 0.2 per unit per year

= 0.01.

Hence $EOQ = (2Rc_o/c_h)^{0.5}$

$= (2 \times 12000 \times 5/0.01)^{0.5}$

= 3464.

2.17 WORKING WITH AN INVENTORY SYSTEM

An important class of simulation problems involves inventory systems. Let us consider a company that sells a single product. Suppose it would like to be familiar with how many items it should have in inventory for each of the next n months, where n is the fixed input parameter. The times between demands are independent and identically distributed (IID) exponential random variables with a mean of 0.1 month. Identically distributed means the inter-arrival time has the

same probability distribution. The demand-size random variate D must be discrete and can be generated as follows:

First divide the unit interval into the continuous subintervals as

NOTES

$$\left.\begin{aligned} C_1 &= \left[0, \frac{1}{6}\right], \\ C_1 &= \left[\frac{1}{6}, \frac{1}{2}\right], \\ C_3 &= \left[\frac{1}{2}, \frac{5}{6}\right], \text{ and} \\ C_4 &= \left[\frac{5}{6}, 1\right] \end{aligned}\right\} \quad ...(32)$$

The width of C_1 is $\frac{1}{6} - 0 = \frac{1}{6}$,

C_2 is $\frac{1}{2} - \frac{1}{6} = \frac{1}{3}$,

C_3 is $\frac{5}{6} - \frac{1}{2} = \frac{1}{3}$, and

C_4 is $1 - \frac{5}{6} = \frac{1}{6}$.

So, the sizes of demands D, are IID random variable with

$$D = \begin{Bmatrix} 1 \text{ with probability } 1/6 \\ 2 \text{ with probability } 1/3 \\ 3 \text{ with probability } 1/3 \\ 4 \text{ with probability } 1/6 \end{Bmatrix} \quad ...(33)$$

At the beginning of each month the company reviews the inventory level and divides how many items to order from its supplier.

When an order is placed, the time required for it to arrive is a random variable that is distributed uniformly over a period and is called delivery lag or lead time.

When a demand occurs, if the inventory level is at least as large as the demand, it is satisfied immediately.

If the demand exceeds the inventory level, the excess of the demand over supply is backlogged indicating shortage and these units of goods are back ordered. In this case, the new inventory level is equal to the old inventory level minus the demand size, resulting in negative inventory level. When an order arrives, it is first used to eliminate as much of the backlog (if any) as possible. The remainder of the order (if any) is added to the inventory. Shortage results in lost sale and loss of goodwill. To avoid shortages, a buffer, or safety, stock should be carried out.

NOTES

Carrying stock in inventory has an associated cost known as opportunity cost *i.e.*, it is having capital tied up in inventory rather than invested elsewhere and not having the funds available for other investment purpose. Another cost is the holding or carrying cost. This includes cost like renting of warehouse, maintenance, insurance, taxes, interest, deterioration, etc. An alternative to carrying high inventory is to make more frequent reviews, and consequently, more frequent purchases; Reordering cost is associates with this.

When a backlog occurs in the company then, backlog cost includes cost of extra record keeping when a backlog exists as well as loss of customer goodwill. Larger inventories decrease the possibilities of shortage. These costs can be traded off in order to minimize the total cost of an inventory system.

If a company order X items then the cost incurred is calculated by considering a setup cost k and investment cost per item ordered i.

$$\text{Cost (C)} = k + iX \qquad ...(34)$$

The lead time or delivery lag is between 0.5 and 1 month. It the company uses a stationary (s, S) policy to decide how much to order, *i.e.*,

$$X = \begin{Bmatrix} S - I & \text{if } I < s \\ 0 & \text{if } I \geq s \end{Bmatrix} \qquad ...(35)$$

Where, I is the inventory level at the beginning of the month.

Let $I(t)$ be the inventory level at time t, can be positive, negative, or zero.

$I^+(t) = \max (I(t), 0)$. Inventory at time t. always positive values as it denotes physical quantities on hand.

$I^-(t) = \max (-I(t), 0)$. When backing log at time t.

The points of time at which I(t) decreases are the ones at which demands occur.

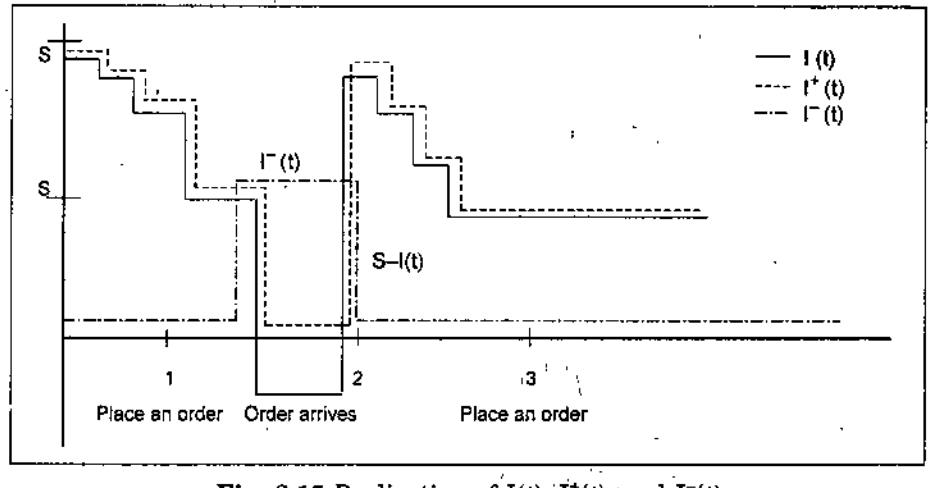

Fig. 2.15 Realization of $I(t)$, $I^+(t)$, and $I^-(t)$

Holding costs are still increased when $I^+(t) = 0$. Our aim is to compare ordering policies and so opportunity cost and holding cost are ignored.

The time-average (per month) is the number of items hold in inventory for the n-month period.

$$\bar{I}^+ = \frac{\int_n^0 I^+(t)dt}{n} \quad ...(36)$$

where $I^+(t)$ is number items held in inventory at time t.

Thus,

average holding cost per months = $h\bar{I}^+$ ...(37)

where h is holding cost of per item per month in inventory. Suppose the company had a backlog then the time-average number of items in backlog is

$$\bar{I}^- = \frac{\int_n^0 I^-(t)dt}{n} \quad ...(38)$$

Thus, the average backlog cost per month is $\pi\ \bar{I}^-$ ...(39)

where π is the backlog cost per item per month in backlog.

Obtain a U (0,1) random Variate, U from the random-number generator. If U falls in C_1, return D = 1, if U falls in C_2, return D = 2; and so on. Since U is uniformly distributed over (0, 1) the probability that U falls in C_1 $\frac{1}{6}$ is, this agrees with desired probability that D = 1.

The delivery lags are uniformly distributed, but not over the unit interval [0, 1]. In general we can generate a random variate distributed uniformly over any interval *(a, b)* by generating a U (0, 1) random number U, and then returning $a + U\ [b - a]$.

Our model of inventory system uses *four* types of events (as shown in table 2.10):

Table 2.10. Event Description

Event Type	*Event Description*
1	Arrival of an order to the company from the supplier.
2	Demand for the product from a customer.
3	End of simulation after n month.
4	Inventory evaluation at the beginning of a month.

End simulation even being exception, all other three events involves state change. For these three events flow chart can be drawn. Assuming initial inventory level as $I(0) = 60$ and no order in outstanding. We simulate the inventory system for $n = 120$ month. Average Total cost per month of following policies is compared.

NOTES

s	20	40	20	20	40	40	40	60	60
S	40	60	80	100	60	80	100	80	100

The order-arrival event is shown in flowchart (Fig. 2.16). When a previous placed order has been arrived from the customer, the following tasks must be carried out:

(*a*) Increase the inventory level by the amount of order.

(*b*) Remove the order arrival from the consideration.

The demand-even is shown in flow chart (fig. 2.17). In this even following *there* tasks must be carried out to process the changes necessary to represent the demand's occurrence:

(*a*) Demand size is generated.

(*b*) Inventory is reduced by his demand size.

(*c*) Time of the next demand is scheduled into the event list.

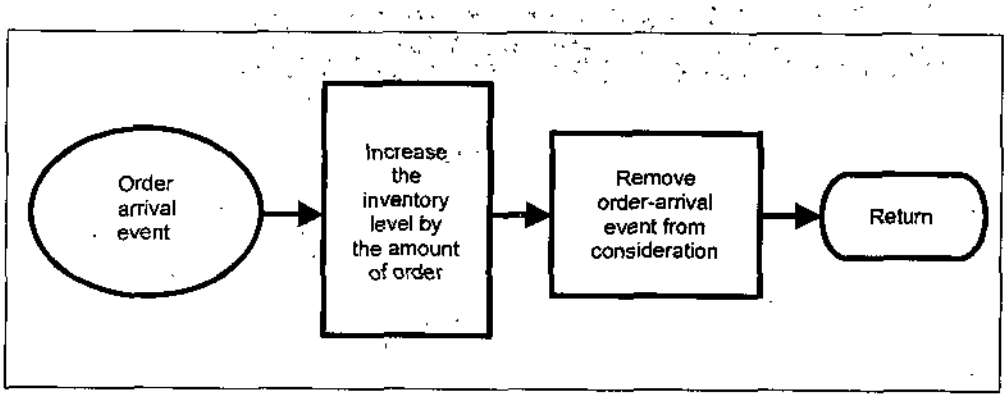

Fig. 2.16 Flowchart of order arrival event, inventory model

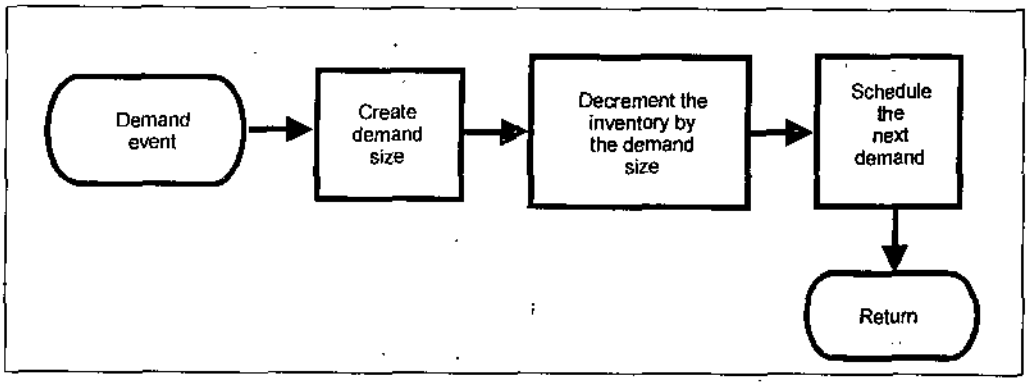

Fig. 2.17 Flowchart of order arrival event, inventory model

Demand event is the place where a backlog can occur *i.e.,* the inventory level might become negative. The inventory-evaluation event takes place at the beginning of each month. In this event the inventory level at a time *t i.e.,* I(*t*) is compared with reorder point *s* as shown in Fig. 2.18.

(*a*) If I(*t*) is at least *s* then

(*i*) No order is placed

(*ii*) Schedule the next evaluation into the event list.

(*b*) If I(*t*) < *S*, then

(*iii*) Place an order for $S - I(t)$ items

(*iv*) Calculate the cost incurred on the order.

(*v*) Schedule the order arrival.

(*vi*) Schedule the next evaluation into the event list.

A simulation model of this inventory system can be easily constructed by stepping time forward in the fixed investment of a month, starting with month and continuing up to month 120. On a typical month, Month *I*, first we check to see if merchandise is due to arrive. If yes, then the existing stock S is increased by the quantity that was ordered. If *D* is the demand of month, and if $D \leq S$, stock at the beginning of the month will be $(S - D)$. If $D > S$, then our stock will be zero, and there will be backlog. In either case we calculate the total cost resulting from current month and add it to the total cost c incurred till last month. Then we determine the inventory on hand plus units on order are greater than reorder point. If not place an order and calculate the amount ordered and the schedule of order arrival. Repeat this procedure for 120 months.

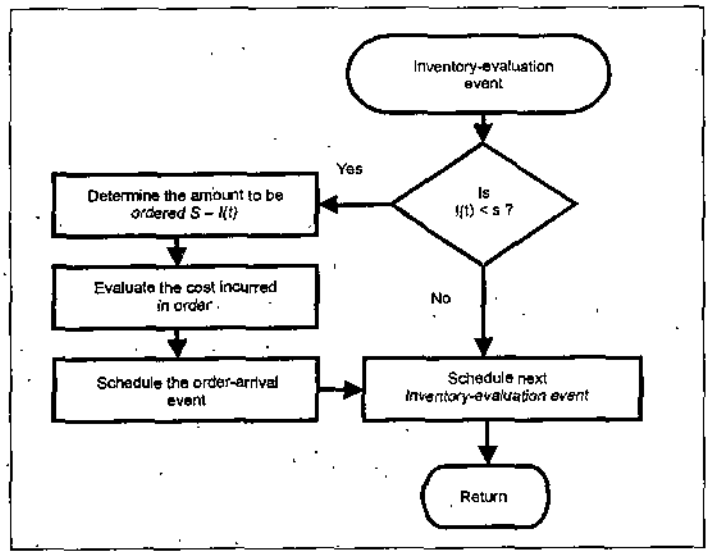

Fig. 2.18 Flowchart for inventory, evaluate on event, inventory model

If the value of I(*t*) is negative, update area under I(*t*), if positive update area under $I^+(t)$, if zero, than no update takes place. When this flow chart is executed with following data the result is obtained:

NOTES

Single Product Inventory System

Initial inventory level	60 items			
Number of demand sizes	4			
Distribution function of demand sizes	0.167	0.500	0.833	1.000
Mean inter-demand time	0.10 months			
Delivery lag range	0.50 to 1.00 month			
Length of the Simulation	120 months			
$K = 32.0$	$i = 3.0$,	$h = 1.0$	$pi = 5.0$	
Number of policies	9			

On combining all these event flow chart into Fig. 2.19 we have,

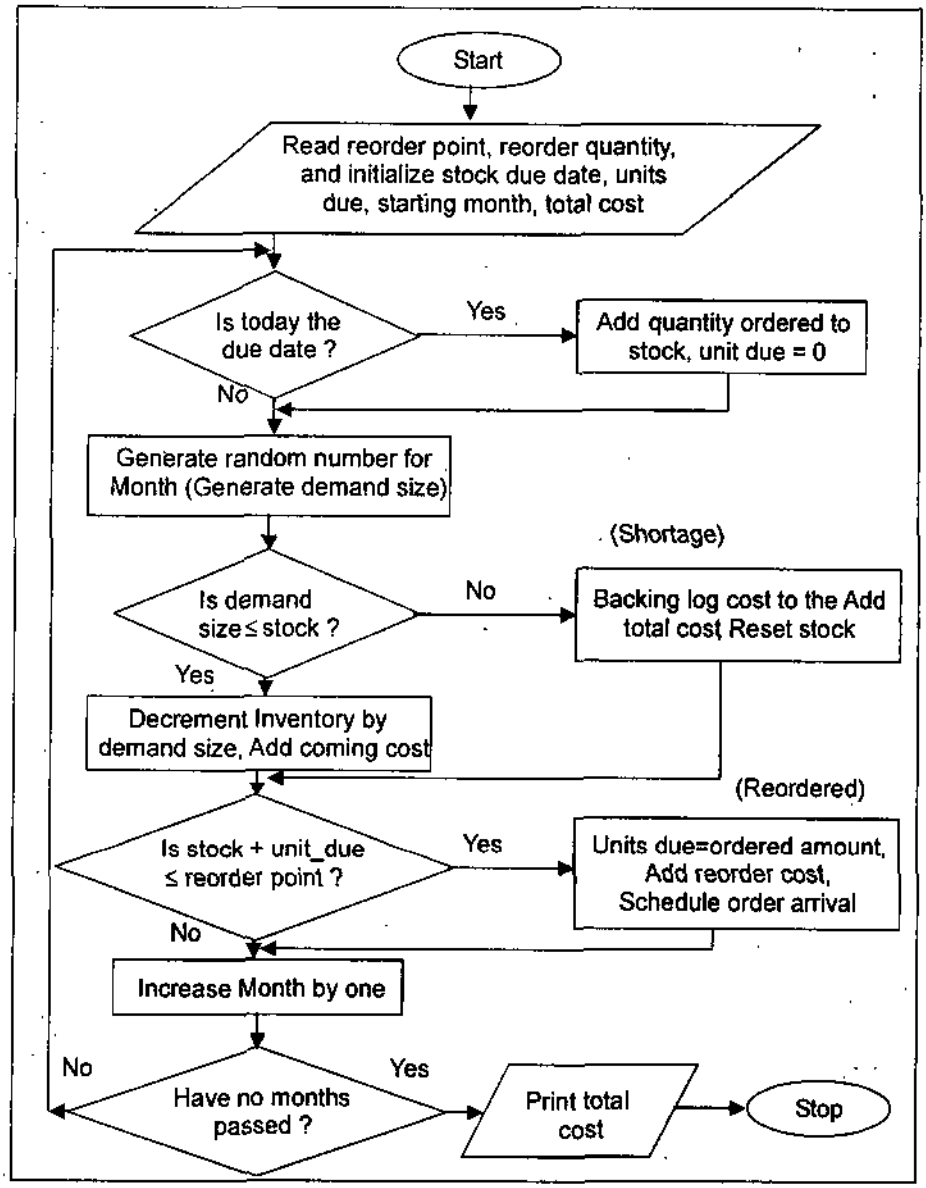

Fig. 2.19 Cost of inventory policy (s, S)

Table 2.11. Cost of Inventory Policy (s, S)

Policy (s, S)	*Average total cost*	*Average ordering cost*	*Average holding cost*	*Average shortage cost*
(20, 40)	126.61	99.26	9.25	18.10
(20, 60)	122.74	91.42	17.39	14.83
(20, 80)	123.86	87.36	26.24	10.26
(20, 100)	125.32	81.37	36.00	7.95
(40, 60)	126.37	91.43	25.99	1.95
(40, 80)	125.46	88.40	35.92	1.14
(40, 100)	132.34	85.62	46.42	1.30
(60, 80)	150.02	103.69	44.02	0.31
(60, 100)	143.20	89.15	53.91	0.24

The three separate components of the average total cost per month are reported to see how they respond individually to changes in s and S, as a possible check on the model and the code. For the example, fixing $s = 20$ and increasing S from 40 to 100.

(*a*) increases holding cost.

(*b*) decreases ordering cost.

(*c*) decreases shortage cost.

Since ordering up to larger values of S implies that these larger order will be placed less frequently, thereby avoiding the fixed ordering cost most often. Fixing $S = 100$ and increasing s from 40 to 80 leads to

(*a*) decrease in shortage cost.

(*b*) increase in holding cost.

(*c*) increase in ordering cost.

Since increment in s translate into less willingness to let the inventory level fall to low values. Since the overall criterion of total cost per month is the sum of three components that move in sometimes different directions in reactions to changes in s and S, we cannot predict even the direction of movement of this criterion without the simulation. Thus we simply look at the values of this criterion, and it would appear that the (60, 80) policy the best, having an average total cost of ₹ 150.02 per month.

In the present context where the length of the simulation is fixed, what we really want to estimate for each policy is the expected average total cost per month for the first 120 months. The numbers are estimates of these expected values, each estimate based on a sample of size I.

NOTES

Since these estimates have large variances, the ordering of them may differ considerably from the ordering of the expected values, which is the desired information. The ordering of the new estimates might also be different. If we re-run the nine simulations using different U(0, 1) random variates, the estimates obtained might differ greatly from those in table 2.11. The ordering of the new estimates might also be different.

2.18 APPLICATIONS

An inventory control system may be used to automate a sales order fulfillment process. Such a system contains a list of order to be filled, and then prompts workers to pick the necessary items, and provides them with packaging and shipping information. Real time inventory control systems use wireless, mobile terminals to record inventory transactions at the moment they occur. A wireless LAN transmits the transaction information to a central database. Physical inventory counting and cycle counting are features of many inventory control systems which can enhance the organization.

The Inventory Control application also provides the ability to monitor inventory transactions, analyze inventory levels, classify inventory for positive control and maintain a history of inventory use as a guide for better planning. Specific features of the Inventory Control application include Inventory gross usage maintenance; Two-step cycle counting; Inventory reorder quantity maintenance including reorder quantities, reorder; Points, and safety stock; Material analysis; Economic order quantity processing; Inventory display (by part number); Inventory analysis; Inventory history display and report; Part number cross reference capability; Inventory history display by part number or transaction types; Stock status report.

2.19 MONTE CARLO SIMULATION

Monte Carlo simulation allows using the most convenient distributions to communicate opinions about the randomness in measurements. For example, a measurement process employing N repetitions of a measurement can be handled explicitly as repeated measurements drawn from a Gaussian, or as a single value drawn from a shifted and scaled Student-t distribution. The former approach may facilitate communication with those who are unfamiliar with the proper use of Student distributions and who would not really understand the latter

approach. The process of Monte Carlo simulation circumvents many complexities of the combination of uncertainty distributions, and only requires combination of the values from the simulated measurements.

It is a simulation in which random statistical sampling techniques are employed such that the result determines estimates for unknown values. Monte Carlo computation (method or technique) was originated during World War II by Ulam and Von Neumann at the Los Alamos Scientific Laboratory. When this approach was applied to problems related to the development of the atomic bombs. In order to design nuclear shields, it was required to know how far neutrons would travel through various materials. The problem was:

(*a*) Too difficult to solve analytically.

(*b*) Too hazardous.

(*c*) Too time consuming to solve experimentally.

Due to above reasons, the experiment was simulated on high-speed computer using random numbers. This technique was called *Monte Carlo technique* based on a gambling-like principle.

Monte Carlo applications are sometimes classified as being simulation and vice versa, presumably because so many simulations involve use of random numbers. Simulation and Monte Carlo are both numerical computational techniques. Simulation applies to dynamic models where as Monte Carlo technique applies to static models.

Deterministic Problems through Random Numbers

Consider a quadrant of a unit circle (see Fig. 2.20).

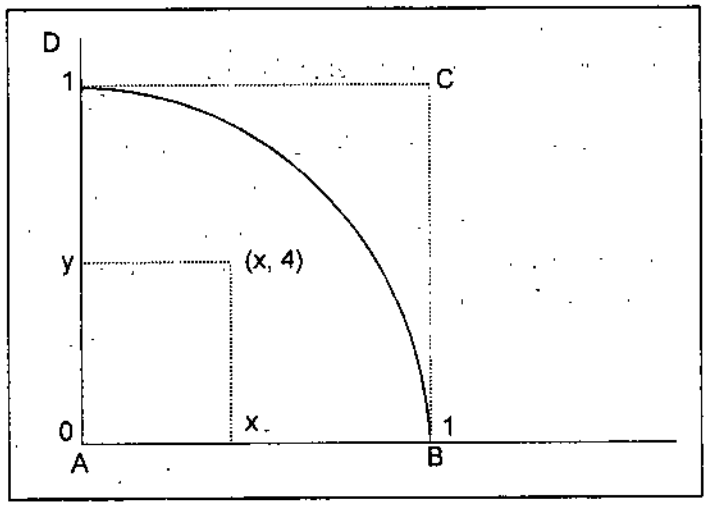

Fig. 2.20 Monte Carlo evaluation of π

NOTES

All the points satisfying the equation 40, lies in this quadrant

$$x^2 + y^2 \leq 1 \text{ where } x, y \geq 0 \quad ...(40)$$

Equation (40) can be rewritten as

$$y \leq \sqrt{1-x^2} \quad ...(41)$$

On generating a pair of uniform random number r_1 and r_2 and in the range (0, 1). The pair (r_1, r_2) is acceptable if equation becomes

$$r_2 \leq \sqrt{1-r_1^2} \quad ...(42)$$

else the pair is rejected. Another pair of uniform random number is generated and tested. Clearly, the entire rejected points lie above the curve and those accepted pair lie below the curve.

If we generate a large number of random pairs (N) and compute the ratio of the number of pairs accepted (n) to those generated, and if N points are used and n of them fall under the curve, than approximately

$$\frac{n}{N} = \int_0^{\pi/2} \frac{f(x)dx}{Area\ of\ rectangle\ ABCD} = \int_0^{\pi/2} \frac{f(x)dx}{1\times 1} = \int_0^{\pi/2} f(x)dx \quad ...(43)$$

The accuracy improves as the number N increase limit 0 to $\pi/2$ as curve is starting from x-axis and ending at y-axis. When it is decided that sufficient points have been taken, the value of the integral is estimated by multiplying $\frac{n}{N}$ by the area of rectangle ABCD. The ratio will approach the area under the curve, which is $\pi/4$. Thus by using random number a completely deterministic problem is solved, called Monte Carlo technique. In this example area under the curve was evaluated through rejection technique.

Monte Carlo Methods

The random number generator picks a new random number every time its called. (The numbers are not really random. Actually the numbers are determined by some rule one after another, but they have good statistical properties and can be used for integration and to simulate randomness.) The numbers are uniformly distributed integers that fall between *0* and *RAND_MAX* which is 2147483648 in this system. To get a random number that is uniformly distributed between zero and one, we compute

```
double x;
. . .
x = rand () / 2147483648.0;
```

Program to compute Pi. Here we use the Monte Carlo method to compute the area of the part inside the unit circle relative to the area of the unit square. ð is four times this fraction.

NOTES

```
/* Monte Carlo Pi montecarlopi.c */
# include <stdio.h>
# include <stdlib.h>
# include <math.h>
# include <time.h>
int main ( void )
{
    double x, y, p ,s=0.0,  ranmaxpo;
    int i, j, k, n = 1000000;
    ranmaxpo = 1.0 + RAND_MAX;

   srand ( (unsigned int)time ( NULL ) );     /* seed for rand is time */
    printf ( "  n\t\t\t\tMonte Carlo Pi MCP-pi\n" );
    for ( k=1; k<= 10; k++ )
    {
       j=0;
       for ( i=1; i <= n; i=i+1 )
       {
          x = rand () / ranmaxpo;
          y = rand () / ranmaxpo;
          if ( x * x  +  y * y  <= 1 )
                j++;
       }
       p = 4.0 * j / n;
       s = s + p;
      printf ( "%4d The approximate for pi is %21.15f %21.15f\n",
k, p, p-M_PI );
    }
    p = s / 10.0;
    printf ( " Average approximate for pi is %21.15f %21.15f\n",
p, p-M_PI );
     printf ( " pi = %21.15f \n", M_PI );
     return EXIT_SUCCESS;
```

```
}
```

Monte-Carlo method for calculating pi

NOTES

```
/* Source code montecarlo.c */
#include <stdio.h>
#include <stdlib.h>
#define RANDOM_MAX 2147483648
int main() {
    int i=0, j=0;
    float x, y;
    while(1) {
      x = (double)random()/(double)RANDOM_MAX;
      y = (double)random()/(double)RANDOM_MAX;
    if ((x*x + y*y) < 1)
      j++;
    i++;
    if (i%1000000 == 1)
    printf("%f\n",4*(float)j/(float)i);
   }
}
```

Monte Carlo Method for the integral of *f(x)*. Here is a program that uses the Monte Carlo method to compute the integral of a function. We compute the area that is both under the curve $0 \le y \le f(x)$ and in the box $a \le x \le b$ and $0 \le y \le c$.

/* For Monte Carlo integral, we assume that the function is given between a <= x <= b We compute the area of the { (x,y) : a <= x <= b and 0 <= y <= min(c,f(x)) } by counting the relative number of random points in the rectangle [a,b]x[0,c] that fall in the set. mo_ca_int.c */

```
# include <stdio.h>
# include <stdlib.h>
# include <math.h>
```

NOTES

```
double f( double x );
int main ( void )
{
    double a, b, c, d, x, y, p, q, r, sum=0.0, ans, ranmaxpo;
    int i, j, k, m=15, n=2000000;
    ranmaxpo = 1.0 + RAND_MAX;
    printf ( " Enter the left, right and upper bounds : ");
    scanf ( "%lf %lf %lf", &a, &b, &c );
    a = fabs ( a );
    b = fabs ( b );
    c = fabs ( c );
    if ( b < a )
    {
          d = a;
          a = b;
          c = d;
    }
    printf ( "\n Monte Carlo Integration of min(f(x),%f) over %f
<= x <= %f\n", c, a, b);

    d = pow( c, 1.0 / 3.0 );
    if ( d >= b)
       ans = ( b * b  + a * a) * ( b * b  -  a * a ) / 4.0;
    else if ( d <= a )
    {
       ans = c * ( b  -  a );
     printf(" Note that here  f(x) > %f  for  %f < x <= %f\n", c, a, b );
    }
    else
    {
     ans = ( d * d  + a * a ) * ( d * d  -  a * a ) / 4.0  + ( b  -  d ) * c;
     printf(" Note that here f(x) > %f  for  %f < x <= %f\n", c, d, b );
```

NOTES

```
    }
    printf( "\n\t n\t Approximate the integral\t\tError\n");
    for(k=1;k<= m; k++)
    {
    j = 0;
    q = c / ranmaxpo;
    r = ( b  -  a ) / ranmaxpo;
        for ( i = 1; i <= n; i = i+1 )
        {
        x = a  +  r * rand ();
        y = q * rand ();
        if ( f(x) > y  )
              j++;
        }
    p = ( b  -  a ) * c * j / n;
    printf ( "%12d%22.15f%22.15f\n", k, p, p  -  ans );
    sum = sum  +  p;
   }
   p = sum / m;
   printf ( "Average int =%21.15f%22.15f\n", p, ans  -  p );
   printf ( "Actual int = %21.15f", ans );
  printf ( " Number of points is %ld\n", (long int)n * (long int)m);
   return EXIT_SUCCESS;
}
double f( double x )
{
      return  x * x * x;
}
```

2.20 DISTRIBUTED LAG MODELS

When the model is large or more complex then the record keeping becomes troublesome. Then it becomes important, the use of computer

and a suitable programming language for simulation. There are some applications of simulation, which are using simple techniques but can be applied without difficulty, even for large models. If all the events occur synchronously, at fixed intervals of time, then the computation remains simple. Distributed lag models are the models which have the following properties:

(*a*) Changing only at fixed intervals of time.

(*b*) Basing current values of the variables on other current values and values that occurred in previous intervals.

These are used extensively in econometric studies where the uniform steps correspond to a time interval, such as a month or a year, over which some economic data are collected. These models consist of linear, algebraic equations. They represent a continuous system, but one in which the data in only available at fixed points in time.

Any variable that appears in the form of the current value and one or more previous intervals is called *lagged variable*. Its value in the previous interval in denoted by attaching the suffix-*n* to the variable, where *n* indicates the interval. 1 denotes the previous interval, 2 denotes the one prior to that and so on.

In distributed lag model an initial set of values is given for all variables and values of the variables at the end of one interval can be derived. Taking these values as the new values of the lagged variables, the values can be derived at the end of second interval and so on. Distributed lag models are conceptually simple and they can be computed by hand or run extensively on computers. The value of computers is its more conventional data-processing capability.

2.21 COBWEB MODELS

The cobweb model explains why prices in certain markets are subject to periodical fluctuations. It is an economic model of cyclical supply and demand in which there is a lag between responses of producers to a change of price. The basic model of supply and demand, the price adjusts so that quantity supplied and the quantities demanded are equal. The precise mechanism that achieves this equilibrium is not always unambiguous.

The cobweb model shows how achieving supply and demand equilibrium might be so automatic if, as seems reasonable the supplier's set the price and the consumer react with a quantity demanded. For some scopes of the demand and supply changes, the equilibrium can be unstable. The *cobweb model* is a classical demonstration that dynamic

NOTES

behavior by economic agents might not converge to a stable equilibrium with supply equal to demand. This application provides two ways to graph the outcome and let you experiment with the key parameter that determines whether the outcome is stable or not.

The cycle will continue to repeat in one of three ways, if the slopes when drawn so that supply was steeper than demand (on price axis), the fluctuation would get wider and wider and fluctuations may become more and more drastic and so a plot of the equilibriums in each period over time would look like an outward spiral (divergent). Alternatively, fluctuation may become less and less drastic and so a plot of the equilibrium in each period over time would look like an inward spiral (convergent). Fluctuations may also remain constant (stable), and so a plot of equilibriums would produce a simple, this scenario is unlikely in the short to medium term. In either of the first two scenarios, the combination of the spiral and the supply and demand curves often look like a cobweb hence the name of the theory.

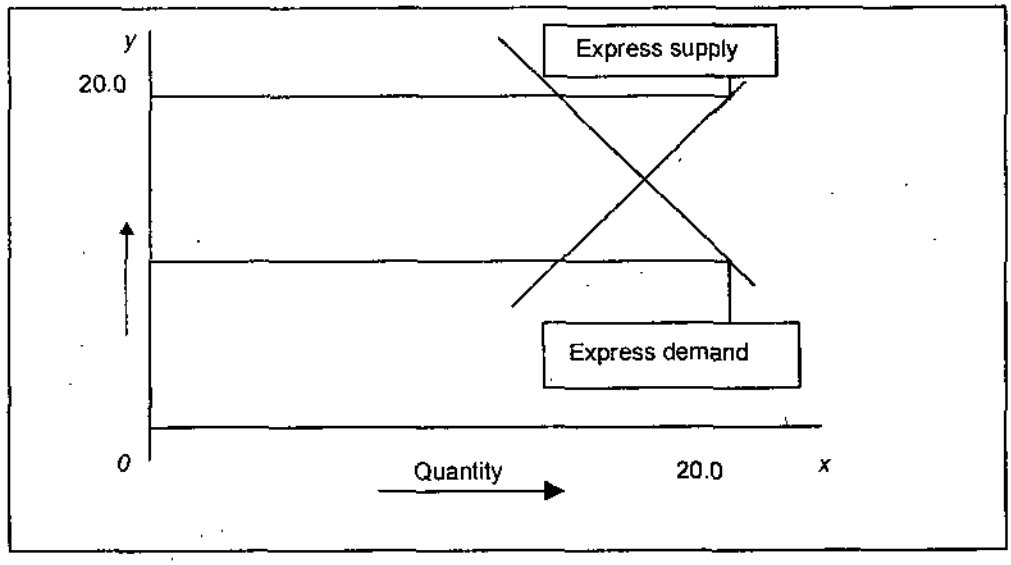

Fig. 2.21 Supply and demand curve.

The inventory-based price model relaxes the assumption that supply must be equal to demand to consider how maintaining an inventory might moderate the possible instability. For example, Retail stores, very often buy an inventory, set a price, and then wait to see what demand might be.

The inventory base-pricing model illustrates that the cobweb might or might not be a realistic challenge to supply and demand equilibrium. Introducing an inventory buffering the difference between supply and demand and letting prices respond to the level of inventory can be

sufficient to eliminate the instability observed for the basic cobweb model. There are two ways to present the cobweb models:

(*a*) **The Traditional Cobweb Model:** This shows the cobweb by alternating between supply and demand.

(*b*) **The Simultaneous Cobweb Model:** This determines supply and demand jointly.

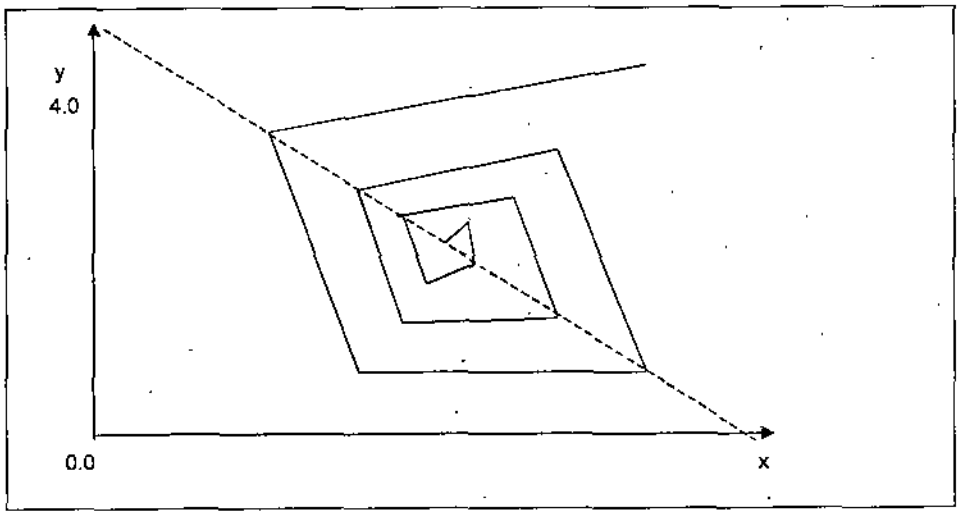

Fig. 2.22 Cobweb of supply and demand.

One criticism of this model is its assumption that producers are extremely short sighted, they are fundamentally unable to judge market conditions of learn from their pricing mistakes that results in surplus or shortfall cycles. This assumption is seen to be unrealistic.

SUMMARY

- Information and communication technology is a vital sector in today's world economy. The future development of this field strongly depends on contributions from mathematics. In the early stages of this development in the design of computer-communication systems, the emphasis was on functionality. In recent years quality of service has become the most important criterion, which is expressed in terms of performance and reliability of the systems in relation to telematics applications. Queueing networks also provide the models for the description of manufacturing systems and for the analysis of their performance and reliability aspects. These economically vital applications of queueing networks make them into objects that are of prime interest.
- Queueing phenomena occur in several real-life situations when resources

NOTES

(machines in a factory, elevators, telephone lines, traffic lights) cannot immediately render the amount or the kind of service required by their users. Similar congestion phenomena also arise at the byte level, in modern data-handling technologies (for example, communication systems, computer networks, etc.); they are typically less visible but their effects at user level are usually very important. Such congestion phenomena are often very effectively studied by mathematical methods from queueing theory. Adopting the abstract terminology from queueing theory, the object of study is formulated as a network of service units with customers requiring services at those units. The nature of the arrival and service processes is usually such that they have to be represented by stochastic processes. Accordingly, queueing theory is an area of applied probability theory and of stochastic operations research.

- With the rapid pace of change in today's retail industry, it is often easy to forget about the importance of getting the basics right. Inventory and how it is managed is a fact of every retailer's life and the common denominator of all retail businesses. The right inventory tracking system makes our life easier and our business more profitable. A good inventory tracking system tells what merchandise is in stock, what is on order, when it will arrive and what we have sold. With such a system, we can plan purchases intelligently and quickly. Researches indicate that stockless inventory has been shown to dramatically reduce hospitals' operating costs while posting annual savings. Despite the savings, and despite the need for hospitals to focus on their bottom line, stockless inventory seems to have lost its momentum.
- Inventory management is an important activity towards ensuring smooth production process. Accounting information plays a key role in inventory management. Just-in-Time (JIT) manufacturing involves purchasing the raw materials and going ahead with the production process as and when the demand arises. JIT is an approach towards minimizing waste and maximizing productivity.
- A large volume of literature on the theory and success stories has been built up on this subject during the past decade. On the other hand, it is known from work on numerical analysis, that numerical methods can introduce instabilities that greatly magnify errors even if the underlying models are stable. To obviate error-induced instabilities, criteria that enable choice of time-step size and other controllable factors are well known for non-distributed simulations. However, the major difference between distributed simulations and their non-distributed counterparts is that control and data are encoded in time stamped messages that travel from one computer to another over a (bandwidth limited) network.

Traditional analyses in the design of numerical methods consider trade-offs between accuracy and speed of computation. However, since distributed messaging requires that continuous quantities be coded into discrete packets and sent discontinuously, it is more appropriate to consider discrete event simulation as a natural means to consider accuracy or bandwidth trade-offs. Recent work has shown that significant reductions of message bandwidth demands (number and size of messages) with controllable error and local computation costs are possible. Finally, the issue of numerical stability in complex simulation is related to the problem of sample path continuity with respect to parameter and timing perturbation.

GLOSSARY

- **Simulation:** Is the process of designing a model of a real system.
- **System:** Is a group of collection of interrelated elements that corporate to accomplish some stated objectives.
- **Model:** Is a representation of a group of objects or ideas in some form other than that of the entity itself.
- **Model translation:** It is formulation of the model in an appropriate simulation language.
- **Experimentation:** Is excuting the simulation to generate the desired data and to preform sensitivity analysis.

REVIEW QUESTIONS

1. Why do we go for simulation? Why is simulation important?
2. When do we use simulations? What must we know before simulation?
3. Explain the differences between simulation and analytical methods.
4. Explain the basic nature of simulation.
5. What are important types of system simulation? Explain continuous system simulation.
6. Write short notes on
   (*i*) Discrete System Simulation
   (*ii*) Real-time Simulation
   (*iii*) Hybrid Simulation
   (*iv*) Social Simulation
   (*v*) Object-oriented Simulation
   (*vi*) On-line Simulation.

NOTES

7. Explain web-based simulation and distributed simulation in detail.
8. Explain the distributed lag models in detail with the help of suitable diagram.
9. Explain cobweb models. In what types of simulation applications it can be used?
10. Explain the role of modeling and simulation in product development process?
11. What do you mean by number of servers in a queueing system?
12. Explain the difference between single server queueing system and general queueing system.
13. Explain the benefits of inventory control in organizations.
14. Explain the features of a good inventory software.

FURTHER READINGS

- '*Modeling and Simulation Concepts*', by Rajinder Kumar, Anil Kumar, Vikesh Kumar, Chandra Shekher Yadav. University Science Press.

UNIT III SYSTEM SIMULATION

NOTES

★ STRUCTURE ★

3.0 LEARNING OBJECTIVES

After going through this unit, you will be able to:

- define continuous and discrete system simulations.
- illustrate analog Vs. digital simulation.
- explain simulation of water reservoir system, servo system and of an autopilot.
- state fixed time-step Vs. Event-to-event model.
- describe generation of random numbers and test for randomness.

NOTES

3.1 INTRODUCTION

Simulation has been applied in many fields, such as aerospace or energy production, but to date it has not seen broad practical application in software engineering. This may be because it is more difficult to accurately model human and organizational behavior than to model physical systems, or it may be that the emphasis on software process is a relatively recent phenomenon. Whatever the reason, this is unfortunate because the rewards from it use are many-fold.

3.2 CONTINUOUS SYSTEM SIMULATION

Let us first discuss, what are continuous events?

Continuous Events. In Continuous events the state variables change continuously as a function of time. for example airplane flight; the state variables like position, velocity change continuously. The distributed lag model is continuous event model and the simulation of such model is continuous system simulation. This model describes how the attribute of the system are related to each other in the form of linear algebraic equations. In general, in continuous system, the relationships describe the rates at which the attributes change, so that the model contains differential equations.

Continuous system simulation typically solves sets of differential equations numerically over time, may involve stochastic elements. Some specialized software available for continuous system simulation. Some discrete-event simulation software can also formulate continuous simulation as well.

Consider the pair of first order ordinary differential equations known as the Lotka-Volterra predator-prey model.

$$y1' = (1 - alpha*y2)*y1$$

$$y2' = (-1 + beta*y1)*y2$$

The functions y1 and y2 measure the sizes of the prey and predator populations respectively. The quadratic cross term accounts for the interactions between the species. Note that the prey population increases when there are no predators, but the predator population decreases when there are no prey. To simulate a system, create a function (or say MATLAB file) that returns a column vector of state derivatives, given state and time values. For this example, we have created a file called LOTKA.M.

```
type lotka

function yp = lotka(t,y)
```

%LOTKA Lotka-Volterra predator-prey model.

yp = diag([1 - .01*y(2), -1 + .02*y(1)])*y;

To simulate the differential equation defined in LOTKA over the interval $0 < t < 15$, invoke ODE23. Use the default relative accuracy of 1e-3 (0.1 percent).

NOTES

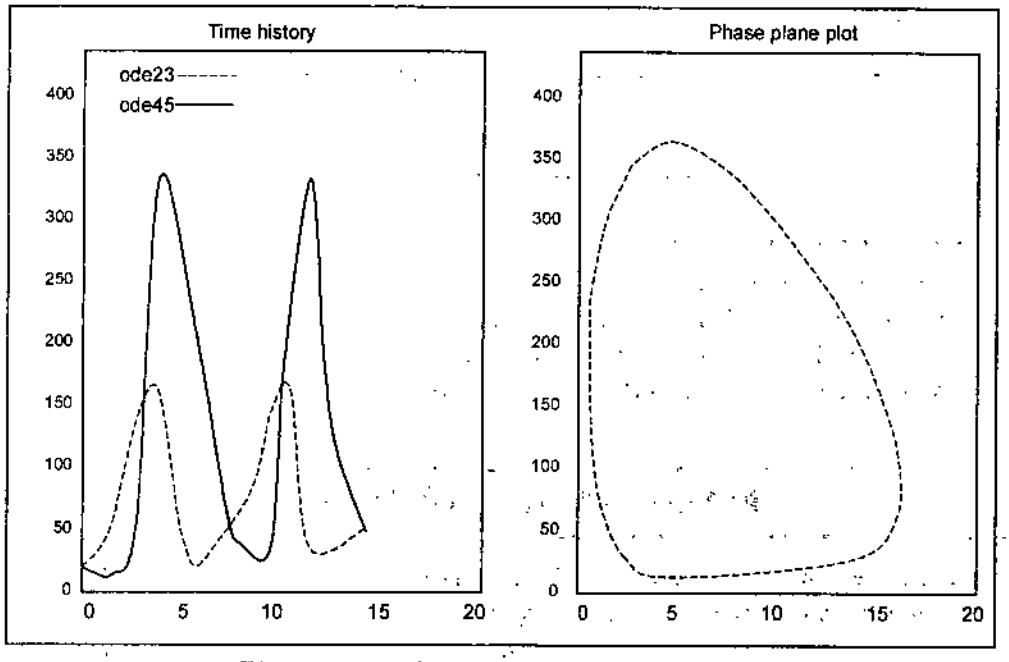

Fig. 3.1 Time history and phase plane plot

% Define initial conditions.

t0 = 0;

tfinal = 15;

y0 = [20 20];

% Simulate the differential equation.

tfinal = tfinal*(1+eps);

[t,y] = ode23('lotka',[t0 tfinal],y0);

%Plot the result of the simulation two different ways.

subplot(1,2,1)

plot(t,y)

title('Time history')

subplot(1,2,2)

plot(y(:,1),y(:,2))

title('Phase plane plot')

Now simulate LOTKA using ODE45, instead of ODE23. ODE45 takes longer at each step, but also takes larger steps. Nevertheless, the output of ODE45 is smooth because by default the solver uses a continuous extension formula to produce output at 4 equally spaced time points in the span of each step taken. The plot compares this result against the previous.

NOTES

```
[T,Y] = ode45('lotka',[t0 tfinal],y0);
subplot(1,1,1)
title('Phase plane plot')
plot(y(:,1),y(:,2),'-',Y(:,1),Y(:,2),'-');
legend('ode23','ode45')
```

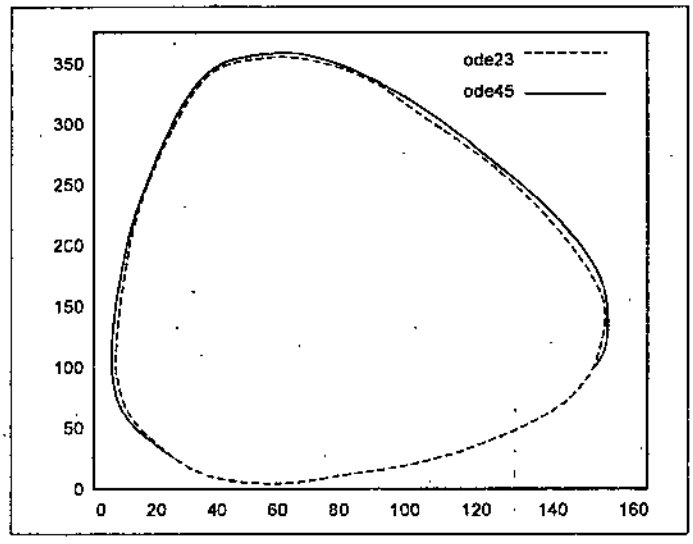

Fig. 3.2 Simulate using ode45 instead of ode23

3.3 DISCRETE SYSTEM SIMULATION

Let us discuss, what are discrete events?

Discrete Event. Between two consecutive events, nothing happens *i.e.*, the graph is horizontal. When the number of events is finite, we call the events "discrete events". In some systems the state changes all the time, not just at the time of some discrete events. For example, the water level in a reservoir with given in and outflows may change all the time. In such cases "continuous simulation" is more appropriate, although discrete event simulation can serve as an approximation.

A discrete-event simulation is one in which the state of a model changes at only a discrete, but possibly random, set of simulated time points. Two or more traffic units often have to be manipulated at one and the same time point. Such simultaneous movement of traffic at a time point is achieved by manipulating units of traffic serially at that time point. This often leads to logical complexities in discrete-event simulation because it raises questions about the order in which two or more units of traffic are to be manipulated at one time point.

Discrete-event simulation models typically have stochastic components that mimic the probabilistic nature of the system under consideration. Successful input modeling requires a close match between the input model and the true underlying probabilistic mechanism associated

NOTES

with the system. Discrete simulation deals with systems whose dynamics can be considered as a sequence of events at discrete time points. The key point of a discrete simulation language is the way it controls the proper sequencing of activities in the model. This is also the way a user must view the world when using the language and a base for classification of discrete simulation languages.

The discrete-event simulation formalism fits the general structure of deterministic, causal systems in classical systems theory. Discrete-event simulation allows for the description of system behavior at two levels. At the lowest level, an atomic discrete-event simulation describes the autonomous behavior of a discrete-event system as a sequence of deterministic transitions between sequential states as well as how it reacts to external input (events) and how it generates output (events). At the higher level, a coupled discrete-event simulation describes a system as a network of coupled components. The components can be atomic discrete-event simulation models or coupled discrete-event simulation in their own right. The connections denote how components influence each other. In particular, output events of one component can become, via a network connection, input events of another component. It is shown in how the discrete-event simulation formalism is closed under coupling: for each coupled discrete-event simulation, a resultant atomic discrete-event simulation can be constructed. As such, any discrete-event simulation model, be it atomic or coupled can be replaced by an equivalent atomic discrete-event simulation. The construction procedure of a resultant atomic discrete-event simulation is also the basis for the implementation of an abstract simulator or solver capable of simulating any discrete-event simulation model. As a coupled discrete-event simulation may have coupled discrete-event simulation components, hierarchical modeling is supported.

Discrete-event systems are dynamic systems, which evolve in time by the occurrence of events at possibly irregular time intervals. Discrete-event systems abound in real-world applications. Examples include traffic systems, flexible manufacturing systems, computer-communications systems, production lines, coherent lifetime systems, and flow networks. Most of these systems can be modeled in terms of discrete-events whose occurrence causes the system to change from one state to another. In designing, analyzing and operating such complex systems, one is interested not only in performance evaluation but also in sensitivity analysis and optimization.

A typical stochastic system has a large number of control parameters that can have a significant impact on performance of the system. To establish a basic knowledge of the behavior of a system under variation of input parameter values and to estimate the relative importance of the input parameters, sensitivity analysis applies small changes to the nominal values of input parameters. For systems simulation, variations

NOTES

of the input parameter values cannot be made infinitely small. The sensitivity of the performance measure with respect to an input parameter is therefore defined as (partial) derivative.

A discrete-event simulation is one, which employs a next-event technique to control the behavior of the model. Many applications of discrete simulation involve queueing systems of one kind or another. The queueing structure may be obvious as in a queue of jobs waiting to be processed on a batch computer or in a stack of aircraft waiting for landing space at an airport.

3.4 ANALOG VS. DIGITAL SIMULATION

We have continuous, dynamic nonlinear system that described by the Van der Waal equation. Simulation program for that system can be described as below.

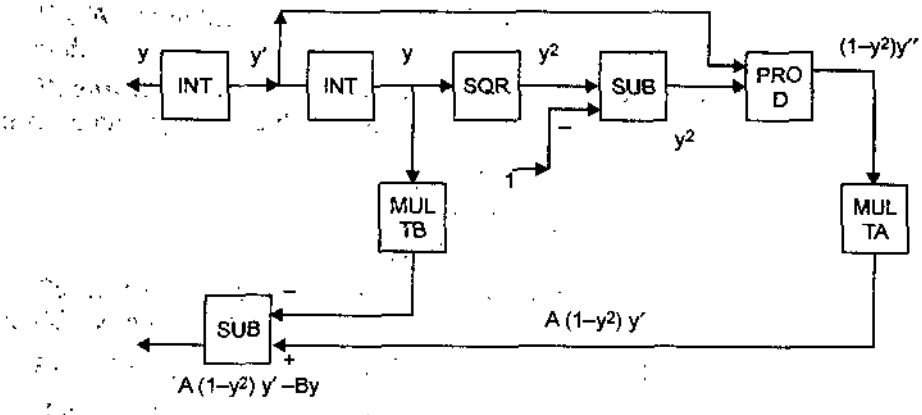

Fig. 3.3 Block-diagram for analog vs. digital simulator

Each block performs mathematical operation. Block INT as integration, SQR squares, PROD takes the products. MULTI multiplies and SUB as subtracts. The blocks may be looked upon as subroutines. The interconnection of the blocks is self-explanatory that the variables y", y', y2 etc., vary with time and must be change every H *i.e.*, Δt seconds. The complete sequence of operations blocks repeated every H seconds if we use Euler integration.

Block Diagram Oriented Languages

Any continuous system numerical solution of different equations as complex simulated by suitably patching together a number of blocks. Different system will vary only in the number of blocks required and their interconnections. Since the same few types of blocks are used and again for simulation of all continuous system, it would be vary to provide a package of standard subroutines to perform the operation

of these blocks. It would be more convenient to have a special programming language which allowed direct implementation of block programs such as the one in Fig. 3.3. A large number of such continuous simulation languages have been designed and implemented. One or more these languages are available at almost every computing system.

Analog Implementation

As each of blocks is actually implemented is immaterial, as long as it assigned task so that the entire set up gives us the state of system as a function of time. These block functions may all be performed by digital computer or by different microprocessors.

A block diagram, as shown in Fig. 3.3 can be simulated on a analog computer, which consists of special-purpose elements as integrators, multipliers, adders, function generation and nonlinear elements. These elements are interconnected to imitate the system under simulation. The variables and their derivatives are represented by means of voltage. So voltages can be monitored continuously at suitable points in the system to get the state of the system as a function of time.

Historically analog computers came into being years before digital computers did, and have played a major role in simulation of continuous dynamic system. But analog computer are giving way to digital computer at increasing pace, they are still in occasional use.

Disadvantages of Analog Computers

(*i*) ***Inadequate Accuracy***: In general the result from a digital simulation is more accurate than that of analog simulation. The accurate analog simulation depends on the accuracy of the components that are 0.01 % to 2% and when accuracy of the components is more than 0.1%, the cost of components increase rapidly. But 1% for a simulation of system with modest complexity is good.

(*ii*) ***Scaling Needed***: The magnitudes of dependent variables are represented in an analog computer by voltages. For a 10 volt analog computer maximum range of voltages is from –10 to +10. All program variables be scaled carefully so that non exceed the maximum voltage, if a variable exceed the maximum voltage then the corresponding amplifier program saturated and results become inaccurate. This magnitude scaling tedious task in an analog simulation as there are many variables and their maximum values are not known in advance.

In addition to magnitude scaling analog computer also require time scaling. The maximum computing time for an analog computer is limited because of drifts affecting the accuracy of the results.

NOTES

If the ratio is 1 then simulation is real time. So there is only one time scale or the task of time scaling is much easier. In digital computer the time scaling is not needed.

(*iii*) ***Hardware Set up Necessary:*** In analog simulation input constants and initial conditions are incorporated by setting up voltages and potentiometers, and then the various elements (amplifiers, multipliers, voltmeters etc) have to be connected on a patch board. Such setting up is not needed in digital computer simulation.

A simulation program on a digital computer can be easily stored for reuse. Availability of various mathematical functions on a digital computer is more advantage. In digital simulation no set up is required or any accuracy test is needed, rapid switching from one simulation to another can be made. This result in more or better machine utilization.

Advantages of Analog Simulation

1. Higher speed of simulation that are useful for many applications.
2. Direct access to and immediate display of the computed results.

They were found to be extremely valuable in analysis and design of numerous engineering systems. Analog computers were used for simulating various continuous dynamic systems which were too difficult to lend themselves to analytic studies. About ten years later, in the mid 1950's when digital computers became commercially available, their advantages as greater accuracy, no need of scaling, greater flexibility etc, over analog computers in simulating complex continuous systems were recognized and they began to be used for this purpose.

A general purpose digital computer was required to have certain minimum hardware facilities if it was to be used successfully for simulating large continuous system. Such as hardware floating point arithmetic, long word length to reduce round off errors and a reasonably large random –access memory for handling a large amount of intermediate data. As the hardware and software of the digital computers became increasingly better and cheaper during the last twenty years, the digital computer began to be used more and more for simulating continuous dynamic systems.

3.5 SIMULATION OF A WATER RESERVOIR SYSTEM

Let us consider the following proposal for constructing a dam across a river to create a reservoir. The reservoir is to be constructed at a specified site. The curve of the projected demand for the water from the reservoir has been determined (from the expected growth pattern

and the seasonal fluctuations). The input to the reservoir is from the river inflow and from the rainfall directly over the reservoir. The output consists of the seepage and evaporation (natural) losses, in addition to the water supplied to meet the projected demand. This system (called a simple run-of-river storage demand system) is represented symbolically in Fig. 3.4.

The amount of seepage loss is not a constant but depends on the volume of the water stored. We have been given a curve showing the seepage loss as a function of volume for the proposed reservoir. Likewise, the evaporation loss depends on the area of the exposed surface and the coefficient of evaporation. We are given another curve showing the surface area as a function of volume as well as the seasonal variation of the coefficient of evaporation. Therefore, for a given volume of water in the reservoir at a particular time of the year we can calculate the two losses.

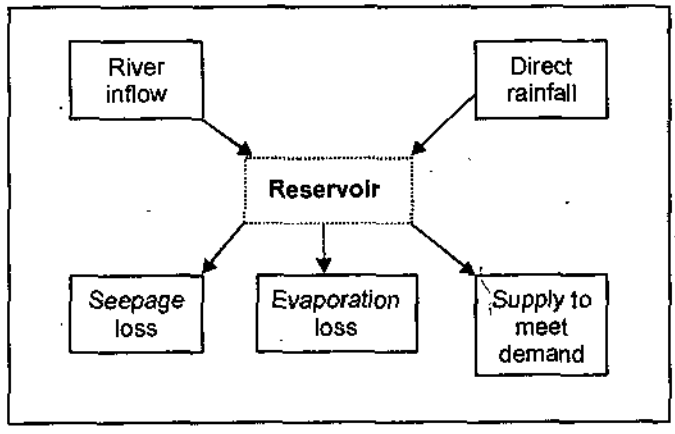

Fig. 3.4 Simple run-of-river storage demand system

In reality no reasonable finite-sized reservoir can provide an absolute guarantee of meeting the demand 100% of the time because the river inflow, the rainfall, the losses, the demand are all random variables. To build such a large dam which will never fail (to meet the demand) through its entire life will generally be uneconomical. Therefore, in practice one determines the reservoir size which will meet the demand with a specified risk of failure (of water shortage). For example, a 2% failure means that once in fifty years the reservoir would become empty before meeting the demand for water. The purpose of the study is to determine the size of the reservoir with a specified risk of failure. There is a single state variable in this system, namely, the volume of water in the reservoir. Since the volume varies continuously with time, we are dealing with a continuous system. It is reasonable to take one month as the basic time interval for the simulation study. Thus, for example, if we wish to simulate the system for 100 years, the simulation run length will be 1200. The simulation will be repeated assuming several different capacities of the reservoir. The output

NOTES

will be in series of ranked shortages for each capacity. The basic procedure, to be repeated for each time step, may be expressed in terms of the following steps:

1. For the current month M of the current year IY determine the total amount of river inflow and the total rainfall directly over the reservoir. Let the sum of two inputs be denoted by VIN (*i.e.*, RAIN + RFLOW).
2. Add the input volume VIN to the volume left over in the reservoir at the end of the last month VOL $(m - 1)$. This gives us the resultant volume, GROSSV = VIN + VOL $(m - 1)$.
3. On the basis of the last month's volume VOL. $(m - 1)$ calculate this month's seepage and evaporation losses and add them as total loss TLOSS = SEEP + EVAP.
4. From the demand curve (stored as a table in the computer memory) determine the demand of water for the current month DEM.
5. If the TLOSS ≥ GROSSV, then the reservoir runs dry without supplying any water and therefore shortage, SHORT = DEM. The volume of the water left at the end of the current month VOL(m) = 0. Spillage SPILL = 0 and go to Step 8; else if TLOSS < GROSSV then the net water volume available to satisfy the demand is VENT = GROSSV – TLOSS.
6. If DEM ≥ VNET the reservoir runs dry and the shortage is given by SHORT = DEM – VNET, and SPILL = 0. Go to Step 8; else if DEM < VNET, then the difference DIFF = VNET – DEM is the water left over.
7. If this leftover water exceeds the capacity CAP of the reservoir there will be a spill over, in other words if DIFF > CAP then SPILL = DIFF – CAP and VOL(m) = CAP; else if DIFF ≤ CAP then SPILL = 0 and VOL(m) = DIFF.
8. Print out SPILL and SHORT for this month, and move to the next month. If the period exceeds the projected simulation length stop, else go to Step 1.

3.6 SIMULATION OF SERVO SYSTEM

A very important application of continuous system simulation is in design and analysis of control systems. Let us study the behavior of a second order non-linear feedback system represented by the following block diagram:

NOTES

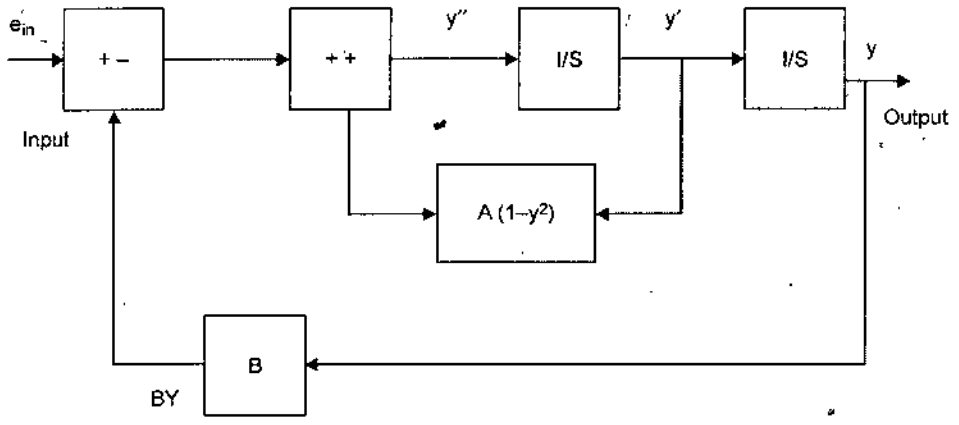

Fig. 3.5 Non-linear second order servo system

This block diagram represents much natural as well as man-made servo system. Some examples are as beating of heart, periodic opening and closing of flowers in response to sunlight, rate of variation of prices, squeaking of door with rusty hinges, dripping of a leaky tap, a neon lamp oscillator etc.

The system of Fig. 3.5 as shown above can also be described by the following differential equation

$$Y'' = A(1 - y^2)y - By + e_{in} \qquad ...(1)$$

where A and B are positive constants

In case of zero input signals, the equation becomes

$$Y'' = A(1 - y^2)y - By \qquad ...(2)$$

This is the well-known Van der Waal non-linear equation. So it can be seen that when the amplitude y is small, the damping term $A(1 - y^2)$ is negative, but when y become large the damping become positive. Thus small-amplitude oscillations will tend to build up, while large-amplitude oscillations will be damped out.

Let us simulate this system. The second order differential equation can be written as a set of two simultaneous equations of first order. We replace Eq. (2) with using variable y^1 in place of y.

$$y_1' = y^2$$
$$y_2' = A(1 - y_1^2)y_2 - By_1$$

To be more specific let constants A = 0.1, B = 1.0 and let the initial conditions be

$$y_1(0) = 1.0$$
$$y_2'(0) = 0.$$

Our equation therefore become

$$y_1 = y_2$$
$$y_2' = 0.1\ (1 - y_1^2)y_2 - y_1$$

An instantaneous description of the state of the system is given by the outputs of the two integrations, *i.e.*, by variables y_1 and y_2. We

NOTES

will use once again the fourth order Runge-Kutta method to obtain the values of y_1 and y_2 as a function of time. We will choose the step-size $\Delta t = H = 0.001$ second and simulate the system for 5 seconds. Thus the number of steps will be N = 5,000. This is too large a number of outputs to be plotted or examined. We will therefore, print out the values of y_1 and y_2 only once every 100 integration steps. This can be implemented by keeping a counter K which is determined by 1 for each integration step. Every time K equals zero, y_1 and y_2 are printed and K is reset to 100.

3.7 SIMULATION OF AN AUTOPILOT

In order to simulate the action of the autopilot, we first construct a mathematical model of the aircraft system. The error signal, ε, has been defined as the difference between the desired heading, or input, θ_i, and the actual heading, or output, θ_o, We therefore have the following identity:

$$\varepsilon = \theta_i - \theta_o \qquad ...(3)$$

We assume the rudder is turned to an angle proportional to the error signal, so that the force changing the aircraft heading is proportional to the error signal. Instead of moving the aircraft sideways, the force applies a torque which will turn the aircraft. The strength of the torque, or turning force, depends on how far back the rudder is placed. However, just as the automobile movement was resisted by a shock absorber, the turning of the aircraft produces a resisting, viscous drag, approximately proportional to the angular velocity of the aircraft. The torque acting on the aircraft can, therefore, be represented by the following equation:

$$\text{Torque} = K\varepsilon - D\theta_0 \qquad ...(4)$$

where K and D are constants. The first term on the right-hand side is the torque produced by the rudder, and the second is the viscous drag.

In case of differential equations, the fundamental law of mechanics is that the acceleration of a body is proportional to the applied force. That was related to linear motion. The same law, however, applies to a turning motion: the coefficient of proportionality is the inertia of the body, denoted by I. Sine the angular acceleration of the aircraft is the second derivative of its heading, the equation of motion is

$$I\theta_o = \text{torque} \qquad ...(5)$$

Substituting from equations (3) and (4), and transposing terms, the resultant equation is

$$I\theta_o + D\theta_o + K\theta_o = K\theta_i \qquad ...(6)$$

If we divide both sides of the equation by I, and make the following substitutions

$$2\xi\omega = \frac{D}{I},\ \omega^2 = \frac{K}{I},$$

the equation of motion relating output to input then takes the following form:

NOTES

$$\theta_o + 2\xi\omega\theta_o + \omega^2\theta_o = \omega^2\theta_i$$

Suppose the aircraft is initially flying a steady course which, by definition, we take to be the zero heading. If it is asked to change to a new heading at time zero, this corresponds to a unit step change of input. Because of the correspondence between the equations of motion we have just noted, the results will be the same, which showed the response of the suspension system to a unit the response will be oscillatory if $\xi < 1$.

To simulate how the autopilot can be designed to modify the aircraft response, it is more convenient to leave the model in the form of there individual equations. Using the variables ERROR and TORQUE to represent the error signal and the applied torque, the equations are:

$$\text{ERROR} = \theta_i - \theta_o$$
$$\text{TORQUE} = K \times \text{ERROR} - D \times \theta_o$$
$$I\theta_o = \text{TORQUE}$$

If we also use the variables HEAD, ANGVEL, and ANGACC to represent the aircraft heading and its first two derivatives, respectively, together with INPUT for the desired heading, the equations can be written

$$\text{ERROR} = \text{INPUT} - \text{HEAD}$$
$$\text{TORQUE} = K \times \text{ERROR} - D \times \text{ANGVEL}$$
$$I \times \text{ANGACC} = \text{TORQUE}$$

A CSMP III program for the system is shown below. Instead of using a step function change of heading, which simply repeats the automobile wheel case, the aircraft is being asked to turn in a circle. The desired heading is then continually increasing at a uniform rate so that

$$\text{INPUT} = A \times \text{TIME}$$

where A is a constant and TIME is a CSMP variable representing the time t^2. The constants K, I, and A have been set to values of 400, 2.00, and 0.0175, respectively. The constant D has been programmed to take different values on five separate runs, so that the damping ratio, ξ, will have the values 0.1, 0.3, 0.7, 1.0, and 2.0. With t expressed in seconds, the equation of motion gives θ_o in radians. The value chosen for A in the simulation makes the input a request for a turn of one degree a second.

```
TITLE AIRCRAFT WITH RATE CONTROL
PARAM D = (5, 656, 16, 968, 39, 592, 56, 56, 113.12)
INPUT = A*TIME
ERROR = INPUT - HEAD
```

NOTES

```
TORUE = K*ERROR – D*ANGVEL
ANGACC = TORQUE/ I
HEAD = INTGRL (0, 0, ANGVEL)
ANGVEL = INTGRAL (0, 0, ANGACC)
CONST I = 2,0, K = 400.0, A = 0. 0175
TIMER DELT = 0.005,  FINTIM = 10.5, PRDEL = 0.05
PRINT HEAD
LABEL HEADING VERSUS TIME
END
```

The results of the runs are plotted in Fig. 3.6. They are given in non-dimensional form by plotting $\omega\theta_0/A$ against Ct. The straight line through the origin is the desired heading, which would be followed by the aircraft if it responded perfectly. However, it can be seen that the aircraft lags behind the desired heading and never catches up so long as the turn request remains in effect. The size of the lag increases as the damping ratio increases. The error signal appears as an electrical signal in the autopilot. It can easily be modified by electrical circuits. The motion of the aircraft can also be measured by instruments giving electrical signals that can be added to the autopilot output. The modifications affect the aircraft response by changing the applied forces. Studies in control theory are concerned with understanding how system behavior can be modified in this way. As an example, suppose it is important to eliminate the lag that builds up as the aircraft's turns. One way this can be done is to add to the error signal a component that is proportional to the integral of the error signal.

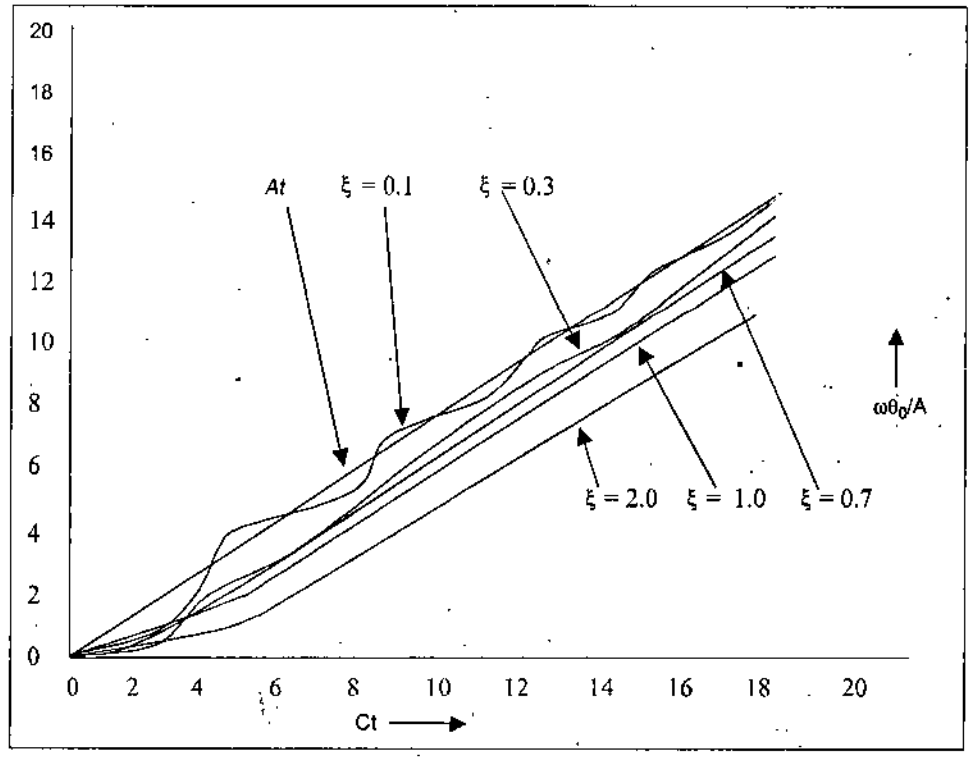

Fig. 3.6 Responses of aircraft flying in a circle with rate control.

The applied force is then as follows:

$$F(t) = K\omega + K_i \int \omega \, dt$$

With this modification, control theory shows that for the system to remain stable, we must have

$$K_1 < \frac{KD}{I}$$

Fig. 3.7 shows the response of the aircraft, under the same conditions as were used before, when this modification is included. In each case, the value of K_1 was Chosen so that $\frac{K_1}{K} = \frac{0.2D}{I}$

The results of Fig. 3.7 show that the second modification, which is called *integral control,* will eliminate the lag when the aircraft is performing a steady turn.

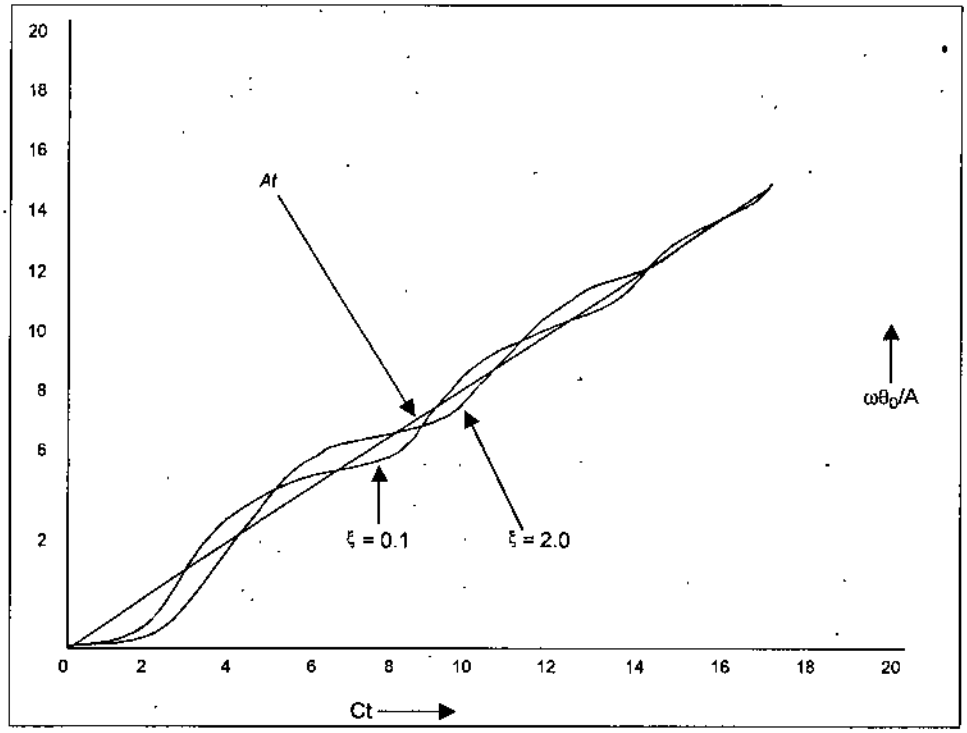

Fig. 3.7 Response of aircraft flying in a circle with integral control.

3.8 FIXED TIME-STEP VS. EVENT-TO-EVENT MODEL

In simulating any dynamic system—continuous or discrete, there must be a mechanism for the flow of time. For we must advance time, keep track of the total elapsed time, determine the state of system at new point in time, and terminate the simulation when the total elapsed time equals or exceed the simulation period. As in continuous system

NOTES

the advance time in small increments of Δt for as long as needed. In simulation of discrete systems, there are two fundamentally different models for moving a system through time: the fixed time-step model and the event-to-event model. In a fixed time step model a "timer" or "clock" is simulated by the computer. This clock is updated by a fixed time interval τ and the system is examined to see if any event has taken place during this time interval. All events that takes place during this period are treated as if they occurred simultaneously at the tail end of this interval. In the next-event simulation model the computer advance time to be occurrence of the next event. It shifts from event to event. The system state does not change in between. Only those points in time are kept track of when something of interest happens to the system.

Differences between Two Models

The differences between two models, for example that we are simulating the dynamics of the population in a fish bowl, start with 10 fish, if we use fixed time-step, model with, say $\tau = 1$ day, in this way we scan the bowl once every 24 hours and any births and deaths takes place are presumed to be during the moment of this period. On the other hand, if we use a next-time event moment then we will first find out when the next event as birth and death is to take place and then advance the clock exactly to that time.

In general, the next event model is preferred because we do not waste any time in scanning those points in time when nothing takes place. The waste is bound to occur if we pick a reasonably small value for τ. On the other hand, if τ is so large that one or more events must take place during each interval then our model becomes unrealistic and may not yield meaningful results. In most simulations of discrete system the next event model is used.

Drawbacks

The next-event model is that usually its implementation (programming for it) turns out to be more complicated than the fixed time-step model.

3.9 GENERATION OF RANDOM NUMBERS

What we call a random number generator (RNG) is actually a program that produces, a deterministic and periodic sequence of numbers, once its initial state is chosen. In generating sequences of random numbers we wish to duplicate, in effect, a game in which a fair wheel of fortune is spun and the outcome of the game is recorded after each

spinning. If the periphery of the wheel of fortune is divided into m equal intervals which are indexed with the integers from I to m, this game produces independent random integers which are uniformly distributed in the 1 to m range.

NOTES

There are basically two ways of obtaining sequences of random numbers for a simulation experiment. The first way is simply to read the random numbers from a list of such numbers which has already been compiled by somebody else. For example, a table with 1 million random digits was published in 1955 by the Rand Corporation. Such tables of random numbers have been recorded on magnetic tape and can thus be read quickly by a digital computer.

The second way (the most common) is to have the computer itself generate a sequence of random numbers. This is accomplished by having the computer execute a short program every time a new random number is needed. This computer program essentially uses the last random number produced, say the $(n-1)$st in the sequence, to produce the next random number, say the nth in the sequence. The specific method employed can be anyone of the congruential methods.

For instance, the mixed congruential method uses the expression

$$x_n = ax_{n-1} = c \text{ (modulo } m) \qquad ...(7)$$

where x_n and x_{n-1} are the nth and $(n-1)$st random numbers in the sequence, respectively, and a, c, and m are suitably chosen positive integers with a $< m$ and $c < m$. The indication "modulo m" means that x_n is the remainder of the division of the quantity $ax_{n-1} + c$ by the number m. For example, if $a = 5$, $x_{n-1} = 7$, $c = 3$, and $m = 16$, we have $ax_{n-1} + c = 38$ and $x_n = 38$ (modulo 16) $= 6$.

How many different numbers can we obtain through equation (7)? Obviously, *at most, m,* that is all the numbers between 0 and $m - 1$. For example, with $a = 5$, $c = 3$, $x, = 7$, and $m = 16$, we obtain $x_1 = 6$ (as we just saw) and then $x_2 = 1$, $x_3 = 8$, $x_4 = 11$, $x_5 = 10$, x_6-5, $x_7 = 12$, $x_8 = 15$, $x_9 = 14$, $x_{10} = 91$, $x_{11} = 01$, $x_{12} = 3$, $x_{13} = 2$, x_{14}-13, and $x_{15} = 4$, in all the 16 different numbers in the range from 0 to 15. What is x_{16} in this sequence? We have $a \cdot x_{15} + c$=23 and x_{16}=23 (modulo m) $= 7$. Thus, the earlier sequence will now be repeated once again with $x_{17} = 6$, $x_{18} = 1$, and so on.

The example above illustrates two interesting points. First, any sequence of random numbers produced through eqn. (7) is cyclical. Each cycle consists of a necessarily finite number of distinct numbers (at most m). After a cycle has been completed, a new cycle identical to the previous one begins. (In our example each cycle consists of 16 distinct numbers.) The finiteness of the cycles will obviously cause problems in a simulation if the length of a cycle is small: a succession of identical short cycles of numbers definitely does not behave as a sequence of

NOTES

independent random numbers (we have $x_{n+\tau}$ where τ is the length of the cycle). However, if m is a very large number and a and c are chosen with sufficient care to make the length of each cycle comparable to the size of m, the finiteness of the cycles is of no practical significance. Consider the case when m is chosen to be equal to 2^b, where b is the number of bits in a binary computer word. This is the choice of m made in computer-based simulations, since 2^b is the total number of integers that can be expressed in binary form with the number of bits available. When $b = 32$, we have $m = 2^{32}$ distinct numbers. With a cycle length in this order of magnitude, it is an entirely academic matter that the same sequence of numbers will be repeated at some future point.

The second observation concerns the meaning of the word random in the case of sequences produced through eqn. (7). Obviously, if we know a, c, and m we can predict perfectly the complete sequence that will follow any initial number x_0. For this reason, the initial number x_0 used to produce some sequence of numbers is called the seed of this sequence. For the same reason, the sequences of numbers produced through eqn. (7) (as well as through other congruential methods) are also called *pseudo-random* numbers. This, however, is also of academic importance as long as the sequences of numbers produced through (eqn. 7) qualify (through passing the appropriate statistical tests) as independent samples from a discrete, uniform probability distribution, that is, as long as the successive numbers appear to an observer to be drawn from a game similar to the spinning wheel game that we described earlier.

In fact, the property of reproducibility for the sequences generated through eqn. (7) is in itself a most desirable one. For, when we wish to perform a simulation experiment under "identical conditions" with some earlier experiment, all we have to do is provide the same seed, x_0, used in the earlier experiment to obtain the same sequence of random (or pseudo-random) phenomena as before.

We have yet to say how the constant positive integers a and c in eqn. 7 are chosen. An unfortunate choice of a and c will unavoidably lead to short cycles of numbers (which are usually anything but uniformly distributed) even for a large m. The branch of mathematics called number theory provides the guidelines for a good choice of a and c. For instance, in the case of digital computers, where, as we have mentioned, we use $m = 2^b$, it can be shown that a should be chosen such that it is equal to I in modulo 4 arithmetic (*i.e.*, $a = 1, 5, 9, 13 \ldots$) and that c should be an odd number. The choice of x_0, the seed, is immaterial in this case as far as the length of the cycle is concerned. The desirable properties of any method for producing sequences of random numbers in a digital computer are:

1. The numbers should appear to be statistically independent of each other (although, strictly speaking, they are perfectly correlated).
2. The numbers should be uniformly distributed over some range.
3. The sequence of numbers should not be self-repeating for any desired length.
4. The random numbers should be producible at very high speed.
5. The method should place minimal requirements on the memory of the computer.
6. Any sequence of random numbers obtained during a given simulation experiment should be reproducible.

The congruential method that we have just described is typical of the techniques used to generate random numbers and possesses all of the properties listed above.

In a practical sense, the important fact is that any modern computer system is preprogrammed to produce sequences of random numbers that possess-at least as all six close approximation of the desirable properties mentioned above. Typically, these numbers are produced by just calling an appropriately named function or subroutine in the computer repertoire (typical names for these mini programs include RAND, RANDU, etc.). All that is required of the programmer is to provide a seed to begin the sequence. The computer subsequently provides automatically the input (*i.e.*, the number x_{n-1} in the mixed congruential method) needed to produce the next number, x_n, in the sequence.

The random numbers produced through these preprogrammed methods are usually presented in the form of numbers uniformly distributed between 0.0 and 1.0. Obviously, the 0 to 1 interval is thus subdivided so finely that, for all practical purposes, it can be assumed that the computer produces statistically independent samples from the continuous uniform probability density function shown on Fig. 3.8. This, too, will be our assumption from here on, and we shall use the expression independent uniformly distributed over [0, 1] random numbers.

Classical uniform random number generators have some major defects, such as, short period length and lack of higher dimension uniformity. However, now a days there is a class of rather complex generators, which is as efficient as the classical generators while enjoy the property of a much longer period and of higher dimension uniformity. Computer programs that generate random numbers use an algorithm. That means if you know the algorithm and the seed values you can predict what numbers will result. Because you can predict the numbers they are not truly random they are pseudorandom. For statistical purposes good pseudorandom numbers generators are good enough.

NOTES

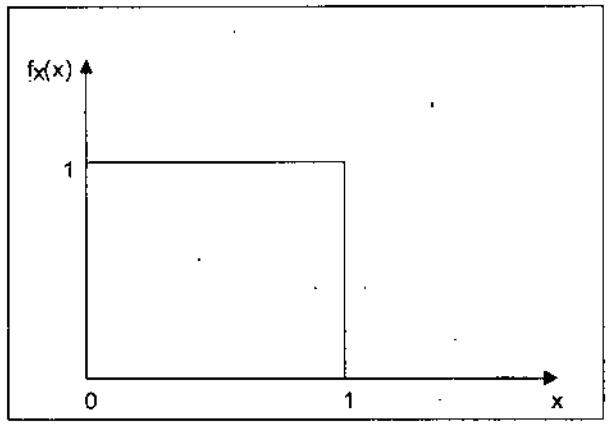

Fig. 3.8 The random variable X with the uniform PDF on [0,1]

real function random()

/* This program returns a pseudo-random numbers with rectangular distribution between 0 and 1. The cycle length is 6.95E+12. IX, IY and IZ should be set to integer values between 1 and 30000 before the first entry. Integer arithmetic up to 30323 is required. */

```
integer ix, iy, iz
common /randc/ ix, iy, iz
ix = 171 * mod(ix, 177) - 2 * (ix / 177)
iy = 172 * mod(iy, 176) - 35 * (iy / 176)
iz = 170 * mod(iz, 178) - 63 * (iz / 178)
if (ix .lt. 0) ix = ix + 30269
if (iy .lt. 0) iy = iy + 30307
if (iz .lt. 0) iz = iz + 30323
```

/* If integer arithmetic up to 5212632 is available, the preceding 6 statements may be replaced by:

```
ix = mod(171 * ix, 30269)
iy = mod(172 * iy, 30307)
iz = mod(170 * iz, 30323)  */
random = mod(float(ix) / 30269. + float(iy) / 30307. + float(iz) / 30323., 1.0)
return
end
real function uniform()
```

/*Generate uniformly distributed random numbers using the 32-bit generator. The cycle length is claimed to be 2.30584E+18 Seeds can be set by calling the routine set_uniform. It is assumed that the Fortran compiler supports long variable names, and integer*4. */

NOTES

```
      integer*4 z, k, s1, s2
      common /unif_seeds/ s1, s2
      save /unif_seeds/
      k = s1 / 53668
      s1 = 40014 * (s1 - k * 53668) - k * 12211

if (s1 .lt. 0) s1 = s1 + 2147483563
      k = s2 / 52774
      s2 = 40692 * (s2 - k * 52774) - k * 3791
      if (s2 .lt. 0) s2 = s2 + 2147483399
      z = s1 - s2
      if (z .lt. 1) z = z + 2147483562
      uniform = z / 2147483563.
      return
      end
      subroutine set_uniform(seed1, seed2)
/* Setting of seeds for the uniform random number generator. */
      integer*4 s1, s2, seed1, seed2
      common /unif_seeds/ s1, s2
      save /unif_seeds/
      s1 = seed1
      s2 = seed2
      return
      end
```

Generation of Uniformly Distributed Random Numbers

A FORTRAN code is given below for a generator of uniform random numbers on [0,1]. Given method is multiplicative linear congruential generator suitable for a 16-bit platform. It combines three simple generators. It is constructed for more efficient use by providing for a sequence of such numbers, LEN in total, to be returned in a single call. A set of three non-zero integer seeds can be supplied, failing which a default set is employed. If supplied, these three seeds, in order, should lie in the ranges [1,32362], [1,31726] and [1,31656] respectively.

```
/* This is the code for portable random number generator for 16-bit
computer. It generates a sequence of LEN pseudo-random numbers,
returned in variable RVEC. */
```

NOTES

```
      SUBROUTINE RANECU (RVEC, LEN)
      DIMENSION RVEC(*)
      SAVE ISEED1, ISEED2, ISEED3
      DATA ISEED1, ISEED2, ISEED3/1234, 5678, 9876/
/* Default values, used if none supplied via an ENTRY call at RECUIN
      */
      DO 100 I = 1, LEN
      K=ISEED1/206
      ISEED1 = 157 * (ISEED1 - K * 206) - K * 21
      IF(ISEED1.LT.0) ISEED1=ISEED1+32363
      K=ISEED2/217
      ISEED2 = 146 * (ISEED2 - K*217) - K* 45
      IF(ISEED2.LT.O) ISEED2=ISEED2+31727
      K=ISEED3/222
      ISEED3 = 142 * (ISEED3 - K *222) - K * 133
      IF(ISEED3.LT.0) ISEED3=ISEED3+31657
      IZ=ISEED1-ISEED2
      IF(IZ.GT.706)IZ = Z - 32362
      IZ = 1Z+ISEED3
      IF(IZ.LT.1)IZ = 1Z + 32362
      RVEC(I)=REAL(IZ) * 3.0899E - 5
100 CONTINUE
      RETURN
      ENTRY RECUIN(IS1, IS2, IS3)
      ISEED1=IS1
      ISEED2=IS2
      ISEED3=IS3
      RETURN
      ENTRY RECUUT(IS1,IS2,IS3)
      IS1=ISEED1
      IS2=ISEED2
      IS3=ISEED3
      RETURN
      END
```

Generation of Non-uniformly Distributed Random Numbers

NOTES

So far we have seen how we can use random number generators to produce numbers in uniform distributions, either discrete, or continuous. However, given a uniform distribution of random numbers in the range [0, 1], we can use easily use this as starting point for more complicated distributions. There are two approaches to generating other distributions, analytical and numeric, using the von Neumanm method.

Analytical Inversion Method. We want to produce random numbers which are distributed according to some arbitrary probability function $f(x)$. In order to be a reasonable probability function, we must have

$$f(x) > 0 \text{ for all } x \quad \frac{n}{N} = \int_0^{\pi/2} \frac{f(x)dx}{Area\ of\ rectangle\ ABCD} = \int_0^{\pi/2} \frac{f(x)dx}{1\times 1} = \int_0^{\pi/2} f(x)dx \quad ...(8)$$

and $$\int_{-\infty}^{\infty} f(x)dx = 1 \quad ...(9)$$

We define the cumulative distribution function:

$$F(x) = \int_{-\infty}^{\infty} f(t)dt \text{ and its inverse function } F^{-1}(y) \text{ i.e., } y = F(x) \leftrightarrow x = F^{-1}(y) \quad ...(10)$$

The analytical inversing method uses $F^{-1}(y)$. If we have a random number, u, generated from a uniform distribution [0, 1], we can transform it to a random number, in the $f(x)$ distribution by calculating $x = F^{-1}(u)$.

This algorithm is simple, but it requires us to calculate the inverse function of the integral of the function we want, which may not be possible analytically. To show how this works in practice, suppose we want random numbers from an exponential decay function. This would be required if, for example, we were modeling radioactive decays.

The exponential decay probability function is given by:

$$f(t) = \frac{1}{\tau} e^{-t/\tau} \text{ for } x \geq 1$$
$$= 0 \quad otherwise \quad ...(11)$$

where 0 is the average lifetime.

Now we may calculate:

$$F(x) = \int_0^x f(t)dt = \int_0^x \frac{1}{\tau} 1 - e^{-t/\tau} = 1 - e^{-t/\tau} \quad ...(12)$$

We can then find the inverse function:

$$u = F(x) = 1 - e^{-x/\tau} \rightarrow x = F^{-1}(u) = -\tau \log(1-u) \quad ...(13)$$

But because u is a random number in the range [0, 1], so as 1-u, this reduces to

$$x = F^{-1}(u) = -\tau \log(u) \quad ...(14)$$

NOTES

Some other common random number functions may also be generated using the using analytically derived formulae.

Von Neumann's acceptance-rejection Method. Von Neumann's method is a numerical method which allows random number distributions from any finite interval to be generated.

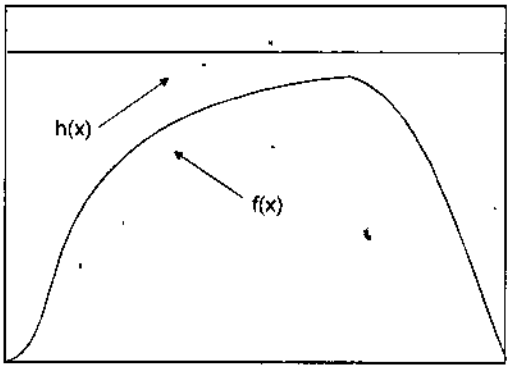

Fig. 3.9 Curve for von Neumann's method

Assuming the required distribution $f(x)$ is known over some range $[a, b]$ and that one can find a second constant function $h(x) = m$ (parallel line to x axix) such that $M > f(x)$ over the whole range. Then one can generate the distribution of random numbers in the $f(x)$ distribution using the following:

(1) Generate a random number, u, from a uniform distribution in the range $[a, b]$

(2). Generate a random number, v, from a uniform distribution in the range $[0, M]$

(3) If $v < f(u)$ we have a *hit* and we accept the random number u else we have a *miss* and we reject the number and go back to step 1.

For example, to generate a random values of in the range of 0 to 5 with a probability distribution of $f(x) = x^2$

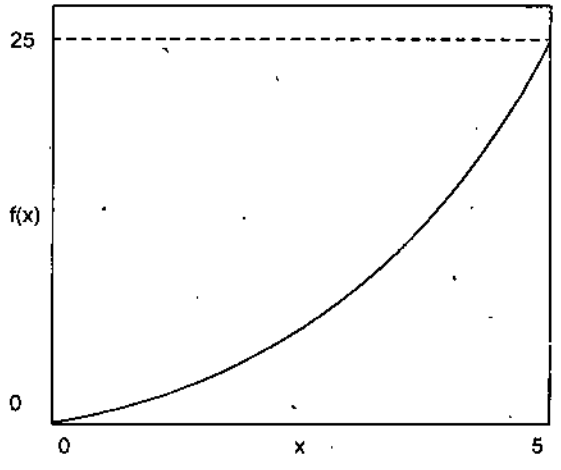

Fig. 3.10 Probability distribution of $f(x) = x^2$

Using von Neumann's method we can generate a single random number using a do-loop in C language:

```
do {
x = 5.0 * (double)rand()/RAND_MAX ;
prob = 25.0*(double)rand()/RAND_MAX;
}
while ( prob > x*x );
```

Here we first generate a value of x from a *uniform* distribution. Then a second random number from [0, 25] is used to accept or reject an event, by comparing this probability with the value of x^2.

NOTES

3.10 TESTS FOR RANDOMNESS

It is obvious that the sequence of numbers generated by multiplicative congruence methods not truly random in the sense that the entire sequence is predetermined and predictable. If the process is started with the same x_0 exactly the same sequence would result. Since m is finite, one of numbers generated will eventually be exactly the same as a previously generated number. Once this happens the subsequence of numbers will also be repeated. For these reasons the name pseudo-random is used for these generators.

For most simulation purposes a random number sequence is adequate if:

1. Its uniformity is assured
2. The successive numbers in the sequence are independent.

For these characteristics the following three tests would occur:

1. **Frequency test:** The frequency test or uniformity test count how often numbers in a given range occur in the sequence to ensure that the numbers are uniformly distributed. These should be no favored numbers that is; no number should occur more frequently than what is expected from chance variation.
2. **Chi-square test:** The Chi-square test is a very important and useful statistical test for determining how well certain observed data fit the theoretically expected data. The testing is performed by first dividing the observed data into k non-overlapping classes; k must be 3 or more as $k = 10$. Then we count O_i the number of time the observed data falls in each class i, for $i = 1, 2...$ k.Next we determine the expected number of occurrences E_i in each class i. Then to measure how far the observed frequency deviates from the expected we compute the chi-square statistic, defined by

$$\text{Chi}^2 = \sum_{i=1}^{k} \frac{(O_i - E_i)}{E_i} \qquad ...(15)$$

NOTES

3. **Independence test:** A sequence may be uniformly and yet be far from being random because the adjacent terms may be related. One of the most efficient method of determining correlation between adjacent terms is the poker test, as supplies we are given five independent decimal digits. We could treat these digits as a hand of poker in a card game and classify accordingly *i.e.,* five of kind (a a a b c); two pairs (a a b b c); one pair (a a b c d) and bust (a b c d e). It is fairly easy to compute the probabilities associated with these seven hands. They are approximately 0.0001, 0.0045, 0.0090, 0.0720, 0.1080, 0.5040, and 0.3024 respectively. If we generated $5n$ random digits we can form n random poker hands and then compare the observed frequencies of these seven types of poker hands with the expected distribution.

SUMMARY

- Simulation has been applied in many fields, such as aerospace or energy production, but to date it has not seen broad practical application in software engineering. This may be because it is more difficult to accurately model human and organizational behavior than to model physical systems, or it may be that the emphasis on software process is a relatively recent phenomenon. Whatever the reason, this is unfortunate because the rewards from it use are many-fold.

  We will discuss here about continuous system simulation, discrete system simulation in detail.

- Simulation can be of major help in pinning down software system requirements early in the product lifecycle, particularly for examining temporal behavior. Simulation can mimic the performance characteristics of software components and their interactions, the effects of time delays and feedbacks, and of finite capacities and resource bottlenecks. Alternate architectures and designs can be evaluated in a safe environment, prior to implementation. In addition, requirements are rarely static, but evolve as experience grows with product development. Thus simulation is not only a valuable tool in defining the initial requirements, but can be used to test out alternate modifications, prior to their implementation. The processes through which the requirements are managed are also critical. However, as far as modeling is concerned, such processes have much in common with other project management processes (e.g., design, development or testing). Finally, a system simulation can be viewed as a component of the requirements and can provide quantitative measures against which the target software system must comply.

- Traditionally, revised or new processes are improved through operational experience. This can be expensive and risky. Simulation can provide

considerable insights into how a process will work, prior to its implementation. These insights can help the process designer assess alternatives, and show that a specific process design performs in a manner that meets expectations. In this way, processes can be pre-tested, and buy-in is more likely obtained from management. Subjective criticisms are less likely, since quantitative simulation of validated models can produce very specific and credible answers to perhaps hostile questions.

- Simulations work because they allow learners to take the valuable knowledge and skills taught in books, e-learning, and the classroom, and practice applying them in an environment that closely approximates the real world. In addition, well-designed custom business simulations evoke emotions, parallel the learner's job, offer specific feedback from multiple sources, encompass predictable and unpredictable events, promote team interaction, and encourage creative decision-making and discovery. Why are simulations so effective? Simulations, unlike any other learning tool, influence the many different ways adults learn effectively. When simulations are used properly to challenge participants to practice multiple skills, they provide an environment in which participants learn through experience. They succeed because there will be something in a simulation that fits each participant's learning style, no matter what that style is.

GLOSSARY

- **Continuous events:** Where the state variables change continuously as a function of time, e.g., airplane flight.
- **Discrete events:** Is the event where state changes all the time. In discrete events number of events is finite.
- **Random number generator (RNG):** Is actually a program that produces a deterministic and periodic sequence of numbers once its initial state is choosen.

REVIEW QUESTIONS

1. Explain Simulation of a water reservoir system.
2. What do you mean by random numbers? What is the importance of random numbers? Write high language program to generate random numbers.
3. Explain the generation of uniformly distributed random numbers.
4. Explain the purpose of generation of non-uniformly distributed random numbers.
5. What are continuous events? Explain the simulation of continuous system.

NOTES

6. Describe Analog vs. Digital Simulation.
7. Write short notes on:
   (a) Discrete system simulation
   (b) Simulation of servo system
   (c) Simulation of an Autopilot
   (d) Test for randomness.

FURTHER READINGS

- *'Modeling and Simulation Concepts'*, by Rajinder Kumar, Anil Kumar, Vikesh Kumar, Chandra Shekher Yadav. University Science Press.

UNIT IV GROWTH MODELS

NOTES

★ STRUCTURE ★

4.0 LEARNING OBJECTIVES

After going through this unit, you will be able to:

- discuss about the system dynamics.
- illustrate the causal loop diagrams.
- explain the exponential growth model.

4.1 INTRODUCTION

System Dynamics is an approach to understanding the behaviour of complex over time. It deals with internal feedback loops and time delays that affect the behaviour of the entire system. What makes using system dynamics different from other approaches? To studying complex system is the use of feedback loops and stocks and flows, the elements help describe how even seemingly simple systems display baffling nonlinearity.

Here, we will discuss about topics related with systems dynamics and exponential growth model in detail.

NOTES

If we plot a graph for x with various values of k and initial values of 1, the graph looks like as shown in Fig. 4.2. From the graph, it can be noted that the population grows indefinitely, for all positive values of k. It can be seen that the population grows faster with greater values of k.

Assume that $k = 0.2$, $x = 2$, and $x_0 = 1$

We have the equation (2) as

$$2 = 1 \times e^{0.21}$$

$$2 = e^{0.21}$$

$$\log 2 = 0.2t$$

$$t = \frac{0.693}{0.2} = 3.46.$$

Let $k = 0.2$, $x = 4$, and $x_0 = 1$, then we have equation (2) as

$$4 = e^{0.2t}$$

$$\log 4 = 0.2t$$

$$t = \frac{\log 4}{0.2} = 6.93.$$

On picking the point the point $x = 2$ at the curve for $k = 0.2$, the corresponding slope at the point 'A' has value 3.46. At point 'B', whose x has become two times *i.e.*, $x = 4$, the slope has become twice as great as at 'A' *i.e.*, 6.93. As the slope is measured as the first order differential coefficient, the growth rate is directly proportional to the current level.

Another way of describing the exponential function is by plotting the logarithms of data against the time and testing whether any particular data represents exponential growth. If the data appears to fall on a straight line, then the growth rate coefficients. It is observed that the logarithm of the variable increases linearly with time. These data can be plotted on semi-logarithmic graph paper where the horizontal lines are placed at logarithmic intervals. Plotting data on such paper is equivalent to taking the logarithm of the data and then plotting on normal linear graph paper. The growth rate coefficient can be estimated by picking two points of the straight line that best fits the data, and then taking the natural logarithm of the ratio of the values. Let the points are x_1 and x_2 at t_1 and t_2 times and $t_2 > t_1$, the result is calculated as:

$$\log \frac{x_2}{x_1} = (t_2 - t_1)k \qquad ...(3)$$

For this equation it is possible to derive the value of k when base 10 logarithm are taken for equation (3), the corresponding result becomes

the curve approaches the limit more slowly the closer it gets, and never actually reaches the limit.

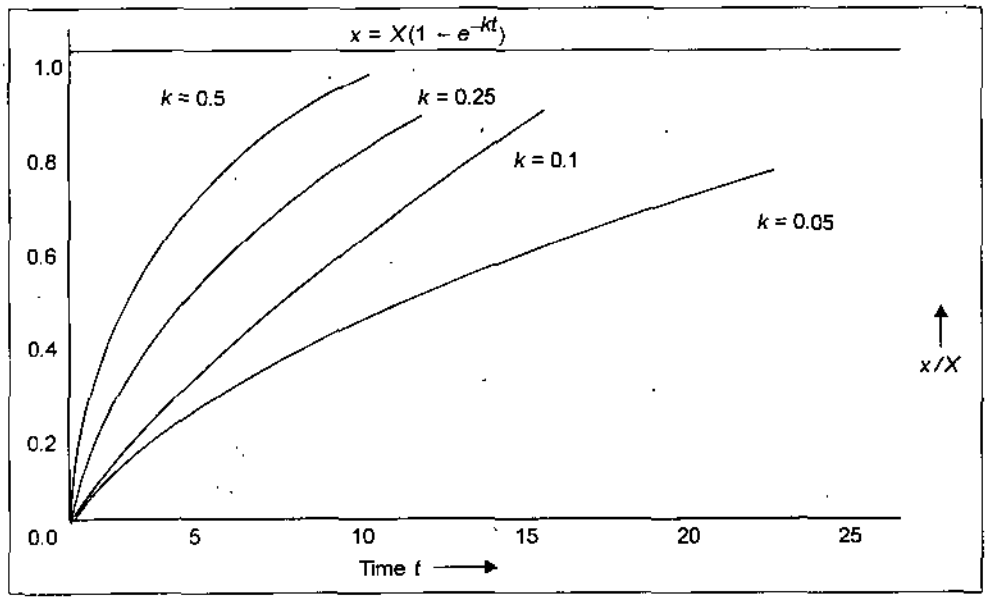

Fig. 4.4 Modified exponential curves

The constant k plays the same role as the growth rate coefficient in exponential growth model. As k increases, the sales grow rapidly. As with the exponential model, the constant is sometimes expressed in the form $1/T$ in which case, it can be interpreted as a time constant.

Logistic Curves

The most unrealistic feature of the modified exponential curve $x = X(-e^{-kt})$, is that it shows the maximum slope (variation) at the beginning. In practice, the slope begins at a low value and initially increases as occurs in the exponential growth model. Eventually the slope begins to decline, making the curve of market growth look more like the modified exponential curve. The result is an S-shaped curve as shown in Fig. 4.5.

Initially the curve turns upwards, looking like the exponential growth model curve. Eventually the curve reaches a point of inflection, where it turns from an increasing to a decreasing slope. From there, the curve resembles a modified exponential curve. For example, when a new product is launched in the market, it is unknown to most. The knowledge of product is spread by advertising and word-of-mouth or by the customers who have actually purchased that product. Under these circumstances, the sales are in proportion to the number of products sold, which is one of the condition governing the exponential growth model. As the number of product sold increases, at a certain point of time the market will be saturated and the condition of the modified exponential model take over.

NOTES

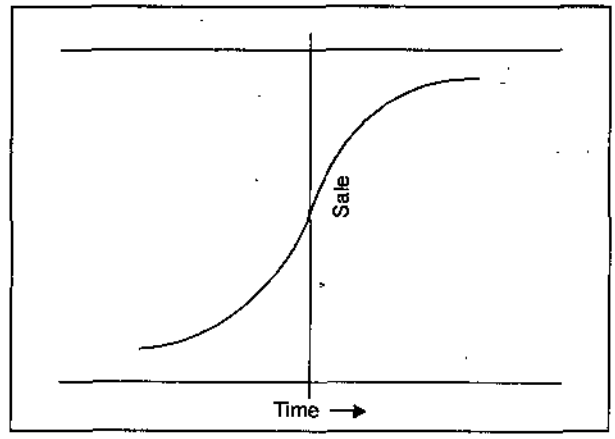

Fig. 4.5 S-shaped growth curve

The logistic function is a combination of the exponential and modified exponential functions, which are used to describe this process mathematically. The differential equation of logistic function is

$$x = kx\ (X - x) \qquad ...(14)$$

x = number of people who buy/purchased the product.

X = number of people who might purchase the product or the market limit.

If x is very much small *i.e.*, the product is newly launched essentially at constant value, X. Then the equation (14) becomes

$$x = kxX \qquad ...(15)$$

This equation is similar to equation (1) of exponential growth model with kX as constant. If $kX = k$ then equation (15) becomes

$$x = k'x \qquad ...(16)$$

When the market is almost saturated, the value of x will be very close to X, the market limit, so that it changes very little with time. So the equation (16) taken the form

$$x = kX\ (X - x) \qquad ...(17)$$

This equation is similar to (equ.6) of modified exponential growth model with KX as a constant. If $KX = k''$ then equation (17) becomes

$$x = K''\ (X - x) \qquad ...(18)$$

This equation (19) is similar to equation (11). The true equation of logistic function is non-linear in nature, as it contains x^2 as a component on simplifying equation (14) we get equation (19).

$$x = kx\ X - kx^2 \qquad ...(19)$$

The solution is complicated and so omitted. The logistic function can also be applicable to population growth corresponds to:

(*a*) The situation in which there are ample resources to support life.

(*b*) The disease being transmitted by contact between infected and uninfected subjects.

The eventual leveling of growth corresponds to:

(*a*) The resources being fully utilized.

(*b*) The number of subjects which are yet uninfected.

Generalization of Growth Models

As we have studied *three* basic growth models:

(*a*) Exponential growth model.

(*b*) Modified exponential growth model.

(*c*) Logistic curves.

Each of these basic models depends upon some coefficients, which are assumed to be constant. But in practice this is not possible. The models can be generalized by removing this assumption. Many other mathematical models could be used as a basis. We are assuming in market model, that the market limit is constant, but it not likely to remain constant. It is going to expand with population growth or fluctuate with economic conditions or respond to both these factors. After finding the best values of the constants to fit in the equation $x = kx(X - x)$ of the model, the graph drawn is confined to the upper portion of a logistic curve.

SUMMARY

- System Dynamics is an approach to understanding the behaviour of complex systems over time. It deals with internal feedback loops and time delays that affect the behaviour of the entire system.
- System dynamics is a methodology and computer simulation modeling technique for framing, understanding, and discussing complex issues and problems.
- The exponential growth model assumes that the rate of growth is proportional to the existing level. This model states an unlimited growth. The exponential growth model cannot be applied to the market model. The market model describes how main items of a product can be sold but there is a limit to the market, if replacement sales are ignored.
- The most unrealistic feature of the modified exponential curve $x = X(-e^{-kt})$, is that it shows the maximum slope (variation) at the beginning. In practice, the slope begins at a low value and initially increases as occurs in the exponential growth model.

NOTES

5.1 INTRODUCTION

Process model simulation allows the activity times of a project to be represented by a variety of distributions and further the resulting project time may also be represented by a variety of distributions. This is a significant improvement over the traditional methods of CPM and PERT. Program Evaluation and Review Technique (PERT) takes the CPM network and adds distributions to represent the activity times of the project. CPM assumes the activity times to be constant, which is not likely in the real world. PERT assumes the activity times of the project to be distributed as Beta distributions and the resulting project time to be a Normal distribution. This is better than assuming them to be constant, but these assumptions are needlessly restrictive.

Complex projects require a series of activities, some of which must be performed sequentially and others that can be performed in parallel with other activities. This collection of series and parallel tasks can be modeled as a network. Network scheduling is typically performed in three phases—network creation, analysis, and development. Although the Critical Path Method (CPM) constitutes a well-established logic in network analysis, human intuition and experience are required for the creation and development of the network. Because of this, a variety of alternative CPM networks can be created in scheduling the same project. The use of the most desirable network can lead to a considerable reduction in the duration of the projects. Accurately identifying activities and linking them in an appropriate manner can achieve this. Many researchers insisted that network scheduling lacks efficiency in scheduling repetitive-unit projects. Because of this, many scheduling methods have been developed to model such types of projects. However, most are not network based and require a large amount of input data, although most leading scheduling software remains network based and field engineers desire network like forms of the schedule.

The *Program Evaluation and Review Technique* (PERT) is a network model that allows for randomness in activity completion times. PERT was developed in the late 1950's for the US Navy's Polaris project having thousands of contractors. It has the potential to reduce both the time and cost required to complete a project.

In 1957 the Critical Path Method (CPM) was developed as a network model for project management. CPM is a deterministic method that uses a fixed time estimate for each activity. While CPM is easy to understand and use, it does not consider the time variations that can have a great impact on the completion time of a complex project.

Gantt chart or Bar chart is the most commonly adopted project-planning tool due to the ease of use and easily understood by project stakeholders,

however this primitive tool falls short in reflecting the interdependencies of WBS (Work Breakdown Structure). Project Network Plan on the other hand is far more effective as its network diagrams can be developed systematically by converting WBS in the forms of nodes and arrows represented in graphical flowcharts for organizations to embrace as project management best practice.

In a project, an activity is a task that must be performed and an event is a milestone marking the completion of one or more activities. Before an activity can begin, all of its predecessor activities must be completed. Project network models represent activities and milestones by arcs/arrows and nodes respectively. PERT originally was an *activity on arc* or *activity on arrow* (AOA) network, in which the activities are represented on the lines and milestones on the nodes. Over time, some people began to use PERT as an *activity on node* (AON) network.

The PERT chart may have multiple pages with many sub-tasks. The following is a very simple example of a PERT diagram:

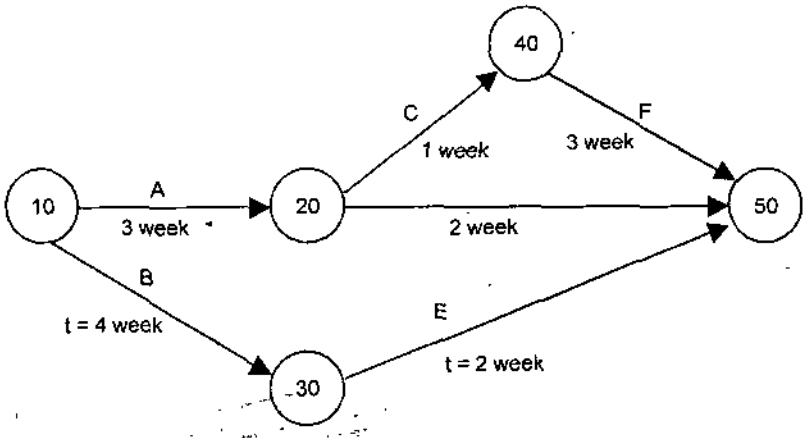

Fig. 5.1 A Pert chart

The milestones generally are numbered so that the ending node of an activity has a higher number than the beginning node. Incrementing the numbers by 10 allows for new ones to be inserted without modifying the numbering of the entire diagram. The activities in the above diagram are labeled with letters along with the expected time required to complete the activity.

The Program Evaluation and Review Technique commonly abbreviated PERT is a model for project management invented by United States Department of Defense's US Navy Special Projects Office in 1958 as part of the Polaris mobile submarine-launched ballistic missile project.

PERT is basically a method to analyze the tasks involved in completing a given project, especially the time needed to complete each task, and identifying the minimum time needed to complete the total project. It was developed in the *i*50*is*, primarily to simplify the planning and scheduling of large and complex projects. It was able to incorporate

uncertainty in the sense that it was possible to schedule a project not knowing precisely the details and durations of all the activities. This technique is used more in R&D-type projects where cost is not a major factor but time is.

NOTES

5.2 PROJECT NETWORK PLAN

The two building blocks of project network plan comprising WBS as input and project network diagrams as output that represented by nodes (boxes) and arrows for a work package. The nodes depict activities whereas arrows show dependencies and project flows. Based on the required work package consists of a series of WBS, a detailed project schedule can be defined with the use of project network diagrams for the entire project.

Project network plan is an effective tool for project planning, scheduling, monitoring and control the work-in-progress. Network diagrams are developed based on a series of WBS in the form of nodes and arrows represented in graphical flowcharts. The nodes display critical information of early start, early finish, late start, late finish, duration and slack time of WBS; whereas the arrows direct project flows from start to finish in a sequential order with interdependencies among nodes and underlining the critical paths. The project network diagrams, apart from being the framework for project information system, can help project managers to make an informed decision on project status, potential risks and constraints. There are two approaches to which project network diagrams can be developed, namely the *activity-on-node* (AON) and *activity-on-arrow* (AOA), the brainchild of Dupont in 1950s to solve its maintenance-scheduling problem with resounding success. AON can be also found in many project software packages that convert Gantt chart into AON once built.

How to Develop an Effective Project Network Plan

Project time management is one of the nine knowledge areas of project management by Project Management Institute (PMI) which underlines the following processes to help develop project time schedule, these processes are also the basis for our project network plan development.

Definition of Activity

To identify specific activities that must be performed to produce the various project deliverables. This can be accomplished by consolidating all inputs on WBS, scope statement, historical information, constraints,

assumptions and expert judgement. Using decomposition and templates as tools and techniques to produce activity list, supporting details and WBS updates.

Sequencing of Activity

To identify and documenting interactivity dependencies. This can be achieved by considering all aspects of activity list, product description, mandatory dependencies, discretionary dependencies, external dependencies and milestones. Tools and techniques may include Precedence Diagram Method (PDM), Arrow Diagram Method (ADM), conditional diagramming methods and network templates to generate project network diagrams (preliminary as noted by author) and activity list updates.

Duration Estimating of Activity

To estimate the number of work periods that will be needed to complete individual activities. This is facilitated with the use of activity list, constraints, assumptions, resource requirements, resource capabilities, historical information, and identified risks. Tools and techniques that will help include the expert judgement, analogous estimating, quantitatively based durations, reserved time to provide the activity duration estimates, basis of estimates and activity list updates.

Development Schedule

To analyze activity sequences, activity durations, and resource requirements to create the project schedule. This can be accomplished with the inputs received on project network diagrams, activity duration estimates, resource requirements, resource pool description, calendars, constraints, assumptions, leads and lags, risk management plan and activity attributes. Tools and techniques that may be useful are the mathematical analysis, duration compression, simulation, resource leveling heuristics, project management software, coding structure to generate project schedule, supporting details, schedule management plan and resource requirement updates.

Schedule Control

Tools and techniques include schedule change control system, performance measurement, additional planning, and project software to define schedule updates, corrective action and lessons learned. Nonetheless, developing an effective project network plan entails a concerted effort from project team members of project manager, engineers and other project stake holders who are the subject experts and knowing the required time taken for the respective WBS of work package with reference to contracted scope and technical complexity.

NOTES

Task *A* is the first task and takes 2 days. When it is done, tasks *B* and *G* can begin. If we follow the task *G* line, it takes 2 days to reach task *H* which takes 5 days. Task *H* leads to the final task, *I*. Total time for following this path is 2 + 2 + 5 + 3 = 12 days. The path would be described as *A*, *B*, *G*, *H*, *I*.

When task *G* began, so did task *B* (with another team of workers). When task *B* finished, after 3 days, there is another opportunity to run some tasks concurrently. So after *B*, tasks *C* and *D* began at the same time.

If we follow task *C*, it takes 1 day to reach task *E*, which leads to the final task *I*. Total time for this path was 2 + 3 + 1 + 4 + 3 = 13 days.

If we followed task *D*, which takes 3 days, it leads to task *F* (also 3 days) before reaching the final task, *I*. Total time for this path is 2 + 3 + 3 + 3 + 3 = 14 days.

Note that tasks *E*, *F* and *H* must all be finished before task *I* can begin.

You will have noticed that there are several paths through from task *A* to task *I*. Each of these paths takes a different amount of time.

Now the question is what is the shortest possible time for the project to take (without leaving any tasks out)?

The answer is 14 days (the longest possible path). It sounds odd that the shortest time is the longest path, but considers another example. You are getting ready for school. At the kitchen table, you have to have breakfast while you finish your maths homework. You have to finish both before you can leave. Breakfast takes 12 minutes. Maths takes 20 minutes. What is the shortest time you would need to leave? *Twenty* minutes, because both tasks must be finished. Just because one task finishes before the other, you cannot leave yet. So in the chart above, the shortest project time would be 14 days.

That is the CRITICAL PATH of the project: the sequence of tasks from beginning to end that takes the longest time. No task on the critical path can take more time without affecting the end date of the project. In other words, none of the tasks on the critical path has any SLACK.

SLACK is the amount of extra time a task can take before it affects a following task. In the breakfast example above, the breakfast could take another eight minutes before it affected the leaving time, so it has eight minutes' slack.

Tasks on the critical path are called CRITICAL TASKS. No critical task can have any slack (by definition).

Before attempting to use these tools, the project's information must

be assembled in a certain way. The following preceding steps can include basic description. The project planning process consists of:

1. Setting the project start date.
2. Setting the project completion date.
3. Selecting the project methodology or project life cycle to be used.
4. Determining the scope of the project in terms of the phases of the selected project methodology or project life cycle.
5. Identifying or selecting the project review methods to be used.
6. Identifying any predetermined interim milestone or other critical dates, which must be met.
7. Listing tasks, by project phase, in the order in which they might be accomplished.
8. Estimating the personnel necessary to accomplish each task.
9. Estimating the personnel available to accomplish each task.
10. Determining skill level necessary to perform each task.
11. Determining task dependencies:

- Which tasks can be done in parallel?
- Which tasks require the completion of other tasks before they can start?
  - project control or review points.
  - performing project cost estimation and cost-benefit analysis.

5.4 PROGRAM EVALUATION AND REVIEW TECHNIQUE (PERT)

A PERT chart is a tool that facilitates decision making; a PERT chart does not make decisions. A PERT chart displays interconnected events (each of which is an important milestone), conventionally represented as numbered circles as in the diagram above. The first draft of a PERT chart will number its events sequentially in 10s (10, 20, 30, etc.) to allow the later insertion of additional events. Two consecutive events in a PERT chart are linked by activities, which are conventionally represented as arrows in the diagram above.

A PERT chart presents a graphic illustration of a project as a network diagram consisting of numbered *nodes* (either circles or rectangles) representing events, or milestones in the project linked by labeled *vectors* (directional lines) representing tasks in the project. The direction

NOTES

of the arrows on the lines indicates the sequence of tasks. In the diagram, for example, the tasks between nodes 1, 2, 4, 8, and 10 must be completed in sequence. These are called *dependent* or *serial* tasks. The tasks between nodes 1 and 2, and nodes 1 and 3 are not dependent on the completion of one to start the other and can be undertaken simultaneously. These tasks are called *parallel* or *concurrent* tasks. Tasks that must be completed in sequence but that do not require resources or completion time are considered to have *event dependency*. These are represented by dotted lines with arrows and are called *dummy activities*. For example, the dashed arrow linking nodes 6 and 9 indicates that the system files must be converted before the user test can take place, but that the resources and time required to prepare for the user test (writing the user manual and user training) are on another path. Numbers on the opposite sides of the vectors indicate the time allotted for the task.

The PERT chart is sometimes preferred over the Gantt chart, another popular project management charting method, because it clearly illustrates task dependencies. On the other hand, the PERT chart can be much more difficult to interpret, especially on complex projects. Frequently, project managers use both techniques.

The events are presented in a logical sequence and no activity can commence until its immediately preceding event is completed. The planner decides which milestones should be PERT events and also decides their "proper" sequence. A PERT chart may have multiple pages with many sub-tasks.

PERT is a variation on Critical Path Analysis (CPA) that takes a slightly more skeptical view of time estimates made for each project stage. To use it, estimate the shortest possible time each activity will take, the most likely length of time, and the longest time that might be taken if the activity takes longer than expected. This helps to bias time estimates away from the unrealistically short time-scales normally assumed.

Key Points:

Critical Path Analysis is an effective and powerful method of assessing:

- What tasks must be carried out?
- Where parallel activity can be performed?
- The shortest time in which you can complete a project.
- Resources needed to execute a project.
- The sequence of activities, scheduling and timings involved.
- Task priorities.
- The most efficient way of shortening time on urgent projects.

An effective Critical Path Analysis can make the difference between success and failure on complex projects. It can be very useful for assessing the importance of problems faced during the implementation of the plan. PERT is a variant of Critical Path Analysis that takes a more skeptical view of the time needed to complete each project stage.

Terminology Related to PERT Charts

***A PERT event*:** is a point that marks the start or completion of one (or more) tasks. It consumes no time, and uses no resources. It marks the completion of one (or more) tasks is not "reached" until all of the activities leading to that event have been completed.

A *predecessor event*: an event (or events) that immediately precedes some other event without any other events intervening. It may be the consequence of more than one activity.

A *successor event*: an event (or events) that immediately follows some other event without any other events intervening. It may be the consequence of more than one activity.

A *PERT activity*: is the actual performance of a task. It consumes time, it requires resources (such as labour, materials, space, machinery), and it can be understood as representing the time, effort, and resources required to move from one event to another. A PERT activity cannot be completed until the event preceding it has occurred.

***Optimistic time* (O):** is the minimum possible time required to accomplish a task, assuming everything proceeds better than is normally expected.

***Pessimistic time* (P):** is the maximum possible time required to accomplish a task, assuming everything goes wrong (but excluding major catastrophes).

***Most likely time* (M):** is the best estimate of the time required to accomplish a task, assuming everything proceeds as normal.

***Expected time* (T_E):** is the best estimate of the time required to accomplish a task, assuming everything proceeds as normal (the implication being that the expected time is the average time the task would require if the task were repeated on a number of occasions over an extended period of time).

$$T_E = (O + 4 \times M + P) / 6$$

***Critical path*:** is the longest possible continuous pathway taken from the initial event to the terminal event. It determines the total calendar time required for the project; and, therefore, any time delays along the critical path will delay the reaching of the terminal event by at least the same amount.

***Lead time*:** is the time by which a *predecessor event* must be completed

in order to allow sufficient time for the activities that must elapse before a specific PERT event is reached to be completed.

Lag time: is the earliest time by which a *successor event* can follow a specific PERT event.

NOTES

Program evaluation and review technique (PERT) charts depict task, duration, and dependency information. Each chart starts with an initiation node from which the first task, or tasks, originates. If multiple tasks begin at the same time, they are all started from the node or branch, or fork out from the starting point. Each task is represented by a line that states its name or other identifier, its duration, the number of people assigned to it, and in some cases the initials of the personnel assigned. The other end of the task line is terminated by another node that identifies the start of another task, or the beginning of any slack time, that is, waiting time between tasks. Each task is connected to its successor tasks in this manner forming a network of nodes and connecting lines. The chart is complete when all final tasks come together at the completion node. When slack time exists between the end of one task and the start of another, the usual method is to draw a broken or dotted line between the end of the first task and the start of the next dependent task.

A PERT chart may have multiple parallel or interconnecting networks of tasks. If the scheduled project has milestones, checkpoints, or review points (all of which are highly recommended in any project schedule), the PERT chart will note that all tasks up to that point terminate at the review node. It should be noted at this point that the project review, approvals, user reviews, and so forth all take time. This time should never be underestimated when drawing up the project plan. It is not unusual for a review to take 1 or 2 weeks. Obtaining management and user approvals may take even longer.

When drawing up the plan, be sure to include tasks for documentation writing, documentation editing, project report writing and editing, and report reproduction. These tasks are usually time-consuming; so do not underestimate how long it will take to complete them.

PERT charts are usually drawn on ruled paper with the horizontal axis indicating time period divisions in days, weeks, months, and so on. Although it is possible to draw a PERT chart for an entire project, the usual practice is to break the plans into smaller, more meaningful parts. This is very helpful if the chart has to be redrawn for any reason, such as skipped or incorrectly estimated tasks.

Many PERT charts terminate at the major review points, such as at the end of the analysis. Many organizations include funding reviews in the projects life cycle. Where this is the case, each chart terminates in the funding review node.

Funding reviews can affect a project in that they may either increase funding, in which case more people have to make available, or they

may decrease funding, in which case fewer people may be available. Obviously more or less people will affect the length of time it takes to complete the project.

Steps in the PERT Planning Process

Identifying Activities and Milestones

The activities are the tasks required to complete the project. The milestones are the events marking the beginning and end of one or more activities. It is helpful to list the tasks in a table that in later steps can be expanded to include information on sequence and duration.

Determining Activity Sequence

This step may be combined with the activity identification step since the activity sequence is obvious for some tasks. Other tasks may require more analysis to determine the exact order in which they must be performed.

Construction of the Network Diagram

Using the activity sequence information, a network diagram can be drawn showing the sequence of the serial and parallel activities. For the original activity-on-arc model, arrowed lines depict the activities and circles depict milestones. If done manually, several drafts may be required to correctly portray the relationships among activities. Software packages simplify this step by automatically converting tabular activity information into a network diagram.

Estimation of Activity Times

Weeks are a commonly used unit of time for activity to be completed, but any consistent unit of time can be used, for example, hours or days. A distinguishing feature of PERT is its ability to deal with uncertainty in activity completion times. For each activity, the model usually includes three time estimates:

- *Optimistic time*—generally the shortest time in which the activity can be completed. It is common practice to specify optimistic times to be three standard deviations from the mean so that there is approximately a 1% chance that the activity will be completed within the optimistic time.
- *Most likely time*—the completion time having the highest probability. Note that this time is different from the *expected time.*
- *Pessimistic time*—the longest time that an activity might require. Three standard deviations from the mean are commonly used for the pessimistic time.

NOTES

PERT assumes a beta probability distribution for the time estimates. Example of a beta probability distribution with minimum value a; maximum value b; and most likely value m, is shown in Fig. 5.3.

A new approach was developed for determining the beta distribution given a, b, and m. For a beta distribution, the expected time for each activity can be approximated using the following weighted average:

Expected time = (Optimistic + 4 × Most likely + Pessimistic) / 6

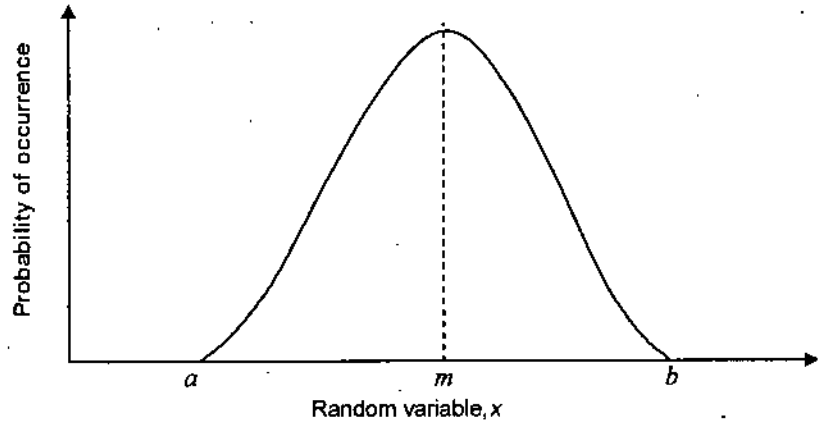

Fig. 5.3 Beta probability distribution

This expected time might be displayed on the network diagram. To calculate the variance for each activity completion time, if three standard deviation times were selected for the optimistic and pessimistic times, then there are six standard deviations between them, so the variance is given by:

$$\tilde{A}^2 = [(\text{Pessimistic}–\text{Optimistic})/6]^2$$

Benefits of PERT

PERT is useful because it provides the following information:

- Expected project completion time.
- Probability of completion before a specified date.
- The critical path activities that directly impact the completion time.
- The activities that have slack time and that can lend resources to critical path activities.
- Activity starting and end dates.

Limitations of PERT

The following are some of PERT's limitations:

- The activity time estimates are somewhat subjective and depend on judgement. In cases where there is little experience in performing an activity, the numbers may be only a guess. In other cases, if the person or group performing the activity estimates the time there may be bias in the estimate.

- Even if the activity times are well estimated, PERT assumes a beta distribution for these time estimates, but the actual distribution may be different.
- Even if the beta distribution assumption holds, PERT assumes that the probability distribution of the project completion time is the same as of the critical path. Because other paths can become the critical path if their associated activities are delayed, PERT consistently underestimates the expected project completion time.

The underestimation of the project completion time due to alternate paths becoming critical is perhaps the most serious of these issues. To overcome this limitation, Monte Carlo simulations can be performed on the network to eliminate this optimistic bias in the expected project completion time.

Implementing PERT

The first step to scheduling the project is to determine the tasks that the project requires and the order in which they must be completed. The order may be easy to record for some tasks while difficult for others. Additionally, the time estimates usually reflect the normal, non-rushed time. Many times, the time required to execute the task can be reduced for an additional cost or a reduction in the quality.

In the following example there are seven tasks, labeled A through G. Some tasks can be done concurrently (A and B) while others cannot be done until their predecessor task is complete (C cannot begin until A is complete). Additionally, each task has three time estimates: the optimistic time estimate (a), the most likely or normal time estimate (m), and the pessimistic time estimate (b). The expected time (T_E) is computed using the formula, $(a + 4m + b)/6$.

Table 5.1. Activities/Task List with their Predecessors

Activity	*Predecessor*	*Optimistic* a	*Normal* m	*Pessimistic* b	T_E $(a + 4m + b)/6$
A	—	2	4	6	4.00
B	—	3	5	9	5.33
C	A	4	5	7	5.17
D	A	4	6	10	6.33
E	B, C	4	5	7	5.17
F	D	3	4	8	4.50
G	E	3	5	8	5.17

NOTES

A network diagram can be created by hand or by using software such as Microsoft Project. There are two types of network diagrams, activity on arrow (AOA) and activity on node (AON). Activity on node diagrams are generally easier to create and interpret. To create an AON diagram, it is recommended (but not necessary) to start with a node named start. This "activity" has a duration of zero (0). Then you draw each activity that does not have a predecessor activity (A and B in this example) and connect them with an arrow. Next, since both C and D list A as a predecessor activity, their nodes are drawn with arrows coming from A. Activity E is listed with B and C as predecessor activities, so node E is drawn with arrows coming from both B and C, signifing that E cannot begin until both B and C have been completed. Activity F has D as a predecessor activity, so an arrow is drawn connecting the activities. Likewise, and arrow is drawn from E to G. Since there are no activities that come after F or G, it is recommended (but again not necessary) to connect them to a node labeled finish.

A network diagram created using Microsoft Project (MSP). Note the critical path is in solid lines.

Note: All times listed are in work days (Monday–Friday, 8 AM to 5 PM with a one hour lunch break). Once this step is complete, one can draw a Gantt chart or a network diagram. Critical path is represented by vertical pattern on horizontal bars and slack is the black lines connected to the non-critical activities (shown by diagonal pattern).

ID	Task	Predecessors	Duration (Days)
1	Start		0.00
2	A	1	4.00
3	B	1	5.33
4	C	2	5.17
5	D	2	6.33
6	E	3, 4	5.17
7	F	5	4.50
8	G	6	5.17
9	Finish	7, 8	0.00

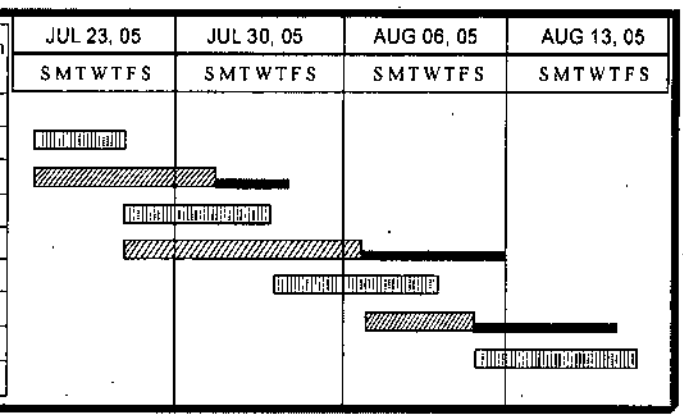

Fig. 5.4 Gantt chart or a Network diagram

Early start	Duration	Early Finish
Task Name		
Late Start	Slack	Late Finish

Fig. 5.5 Task name types in PERT

NOTES

A node like this one can be used to display the activity name, duration, ES, EF, LS, LF, and slack. By itself, the network diagram pictured above does not give much more information than a Gantt chart; however, it can be expanded to display more information. The most common information shown is:

1. The activity name
2. The normal duration time
3. The early start time (ES)
4. The early finish time (EF)
5. The late start time (LS)
6. The late finish time (LF)
7. The slack

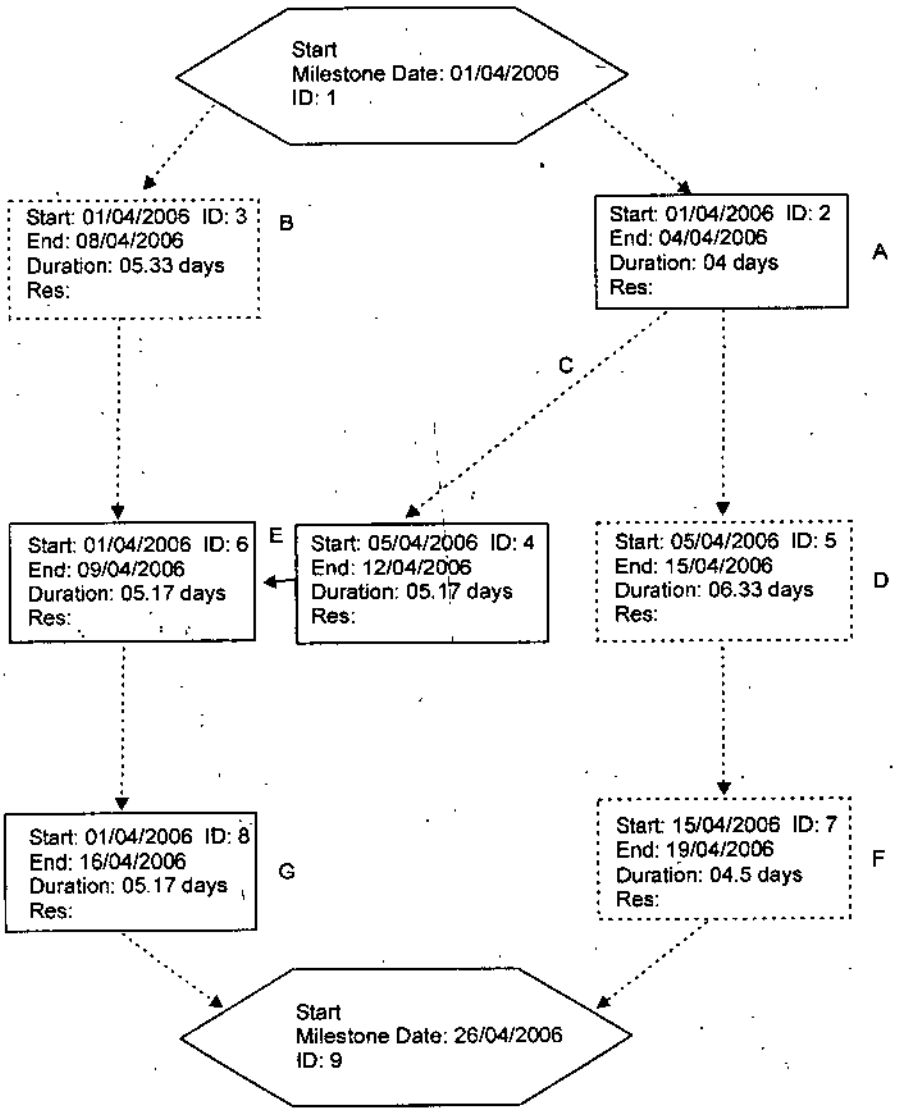

Fig. 5.6 Task flow diagram of above network diagram

NOTES

In order to determine this information it is assumed that the activities and normal duration times are given. The first step is to determine the ES and EF. The ES is defined as the maximum EF of all predecessor activities, unless the activity in question is the first activity, which the ES is zero (0). The EF is the ES plus the task duration (EF = ES + duration).

1. The ES for start is zero since it is the first activity. Since the duration is zero, the EF is also zero. This EF is used as the ES for A and B.
2. The ES for A is zero. The duration (4 work days) is added to the ES to get an EF of four. This EF is used as the ES for C and D.
3. The ES for B is zero. The duration (5.33 work days) is added to the ES to get an EF of 5.33.
4. The ES for C is four. The duration (5.17 work days) is added to the ES to get an EF of 9.17.
5. The ES for D is four. The duration (6.33 work days) is added to the ES to get an EF of 10.33. This EF is used as the ES for F.
6. The ES for E is the greatest EF of its predecessor activities (B and C). Since B has an EF of 5.33 and C has an EF of 9.17, the ES of E is 9.17. The duration (5.17 work days) is added to the ES to get an EF of 14.34. This EF is used as the ES for G.
7. The ES for F is 10.33. The duration (4.5 work days) is added to the ES to get an EF of 14.83.
8. The ES for G is 14.34. The duration (5.17 work days) is added to the ES to get an EF of 19.51.
9. The ES for finish is the greatest EF of its predecessor activities (F and G). Since F has an EF of 14.83 and G has an EF of 19.51, the ES of finish is 19.51. Finish is a milestone (and therefore has duration of zero), so the EF is also 19.51.

Barring any unforseen events, the project should take 19.51 work days to complete. The next step is to determine the late start (LS) and late finish (LF) of each activity. This will eventually show if there are activities that have slack. The LF is defined as the minimum LS of all successor activities, unless the activity is the last activity, for which the LF equals the EF. The LS is the LF minus the task duration (LS = LF–duration).

- The LF for finish is equal to the EF (19.51 work days) since it is the last activity in the project. Since the duration is zero, the LS is also 19.51 work days. This will be used as the LF for F and G.

- The LF for G is 19.51 work days. The duration (5.17 work days) is subtracted from the LF to get a LS of 14.34 work days. This will be used as the LF for E.
- The LF for F is 19.51 work days. The duration (4.5 work days) is subtracted from the LF to get a LS of 15.01 work days. This will be used as the LF for D.
- The LF for E is 14.34 work days. The duration (5.17 work days) is subtracted from the LF to get a LS of 9.17 work days. This will be used as the LF for B and C.
- The LF for D is 15.01 work days. The duration (6.33 work days) is subtracted from the LF to get a LS of 8.68 work days.
- The LF for C is 9.17 work days. The duration (5.17 work days) is subtracted from the LF to get a LS of 4 work days.
- The LF for B is 9.17 work days. The duration (5.33 work days) is subtracted from the LF to get a LS of 3.84 work days.
- The LF for A is the minimum LS of its successor activities. Since C has a LS of 4 work days and D has a LS of 8.68 work days, the LF for A is 4 work days. The duration (4 work days) is subtracted from the LF to get a LS of 0 work days.
- The LF for start is the minimum LS of its successor activities. Since A has a LS of 0 work days and B has a LS of 3.84 work days, the LS is 0 work days.

The next step is to determine the critical path and if any activities have slack. The critical path is the path that takes the longest to complete. To determine the path times, add the task durations for all available paths. Activities that have slack can be delayed without changing the overall time of the project. Slack is computed in one of two ways, slack = LF—EF or slack = LS—ES. Activities that are on the critical path have a slack of zero (0).

- The duration of path A-D-F is 14.83 work days.
- The duration of path A-C-E-G is 19.51 work days.
- The duration of path B-E-G is 15.67 work days.

The critical path is A-C-E-G and the critical time is 19.51 work days. It is important to note that there can be more than one critical path (in a project more complex than this example) or the critical path can change. For example, let us say that activities D and F take their pessimistic (*b*) times to complete instead of their expected (TE) times. The critical path is now A-D-F and the critical time is 22 work days. On the other hand, if activity C can be crashed to one work day, the path time for A-C-E-G is reduced to 15.34 work days, which is slightly less than the time of the new critical path, beg (15.67 work days). Assuming these scenarios do not happen, the slack for each activity can now be determined.

NOTES

- Start and finish are milestones and by definition have no duration, therefore they can have no slack (0 work days).
- The activities on the critical path by definition have a slack of zero; however, it is always a good idea to check the math anyway when drawing by hand.
- $LF_A - EF_A = 4 - 4 = 0$
- $LF_C - EF_C = 9.17 - 9.17 = 0$
- $LF_E - EF_E = 14.34 - 14.34 = 0$
- $LF_G - EF_G = 19.51 - 19.51 = 0$
- Activity B has a LF of 9.17 and a EF of 5.33, so the slack is 3.84 work days.
- Activity D has a LF of 15.01 and a EF of 10.33, so the slack is 4.68 work days.
- Activity F has a LF of 19.51 and a EF of 14.83, so the slack is 3.84 work days.

Therefore, activity B can be delayed almost 4 work days without delaying the project. Likewise, activity D or activity F can be delayed 4.68 work days without delaying the project (alternatively, D and F can be delayed 2.34 work days each).

5.5 CRITICAL PATH METHOD (CPM)

The CPM was developed in the 1950s by DuPont, and was first used in missile-defense construction projects. Since that time, the CPM has been adapted to other fields including hardware and software product research and development. Various computer programs are available to help project managers use the CPM.

The essential technique for using CPM is to construct a model of the project that includes the following:

1. A list of all activities required completing the project (also known as Work breakdown structure),
2. The time (duration) that each activity will take to completion, and
3. The dependencies between the activities.

Using these values, CPM calculates the starting and ending times for each activity, determines which activities are critical to the completion of a project (called the critical path), and reveals those activities with "float time" (are less critical). In project management, a critical path is the sequence of project network activities with the longest overall duration, determining the shortest time possible to complete

the project. Any delay of an activity on the critical path directly impacts the planned project completion date (*i.e.*, there is no float on the critical path). A project can have several, parallel critical paths. An additional parallel path through the network with the total durations shorter than the critical path is called a sub-critical or non-critical path.

These results allow managers to prioritize activities for the effective management of project completion. Originally, the critical path method considered only logical dependencies among terminal elements. Since then, it has been expanded to allow for the inclusion of resources related to each activity. This capability allows for the exploration of a related concept called the critical chain, which determines project duration based upon both time and resource dependencies.

Since project schedules change on a regular basis, CPM allows continuous monitoring of the schedule, allows the project manager to track the critical activities, and ensures that non-critical activities do not interfere with the critical ones. In addition, the method can easily incorporate the concepts of stochastic predictions, using the Program Evaluation and Review Technique.

Currently, there are several software solutions available in industry today that use the CPM method of scheduling. However, the method was developed and used (for decades) without the aid of computers (with pencil and paper).

A Critical Path Method is a project management tool used to formulate a time frame for a project in order to determine where potential delays are most likely to occur. The process includes a step-by-step process that provides the developer with a visual representation of potential bottlenecks throughout the course of the project.

The CPM was originally designed as a method of organizing and tracking the numerous activities regarding the Polaris missile defense program. However, a CPM is useful with many projects and makes the planning process easier.

The critical path method (CPM) is a step-by-step technique for process planning that defines critical and non-critical tasks with the goal of preventing time-frame problems and process bottlenecks. The CPM is ideally suited to projects consisting of numerous activities that interact in a complex manner. In applying the CPM, there are several steps that can be summarized as follows:

- Define the required tasks and put them down in an ordered (sequenced) list.
- Create a flowchart or other diagram showing each task in relation to the others.
- Identify the critical and non-critical relationships (paths) among tasks.

NOTES

- Determine the expected completion or execution time for each task.
- Locate or devise alternatives (backups) for the most critical paths.

CPM Charts

Critical Path Method (CPM) charts are similar to PERT charts and are sometimes known as PERT/CPM. In a CPM chart, the critical path is indicated. A critical path consists that set of dependent tasks (each dependent on the preceding one), which together take the longest time to complete. Although it is not normally done, a CPM chart can define multiple equally critical paths. Tasks, which fall on the critical path, should be noted in some way, so that they may be given special attention. One way is to draw critical path tasks with a double line instead of a single line.

Tasks, which fall on the critical path, should receive special attention by both the project manager and the personnel assigned to them. The critical path for any given method may shift as the project progresses; this can happen when tasks are completed either behind or ahead of schedule, causing other tasks which may still be on schedule to fall on the new critical path.

By completing a CPM the following will be known:

1. The total time to complete the project.
2. The scheduled start and finish dates foe each task pertaining to the projects completion.
3. The tasks that are "critical" to the project and must be completed exactly as scheduled.
4. The "slack" time available in non-critical tasks, as well as how long they can be delayed before they affect the project finish date.

Creating a CPM

Creating a CPM can be quite easy when these simple steps are followed:

1. List the activities to be considered in approximate order.
2. Number the events, estimate the time required for each, and determine the antecedents.
3. Arrange the events in a CPM diagram showing the numbered events, time required, and antecedent relationships with connecting lines.
4. Determine the *earliest* times for starting and ending the events.
5. Determine the *latest* times for starting the events.
6. Determine the critical path.

Step 1: List of Activities

First of all for each activity, show the earliest start date, estimated length of time it will take, and whether it is parallel or sequential. If tasks are sequential, show which stage they depend on. The chart representation in tabular form is shown in table 5.2 (see below):

Table 5.2. Task List: Planning a Custom-written Computer Project

S.N.	*Task*	*Possible start*	*Length*	*Type*	*Dependent*
1.	High level analysis	Week 1	5 days	Sequential	
2.	Selection of hardware platform	Week 1	1 day	Sequential	1
3.	Installation and commissioning of hardware	Week 3	2 weeks	Parallel	2
4.	Detailed analysis of core modules	Week 1	2 weeks	Sequential	1
5.	Detailed analysis of supporting utilities	Week 1	2 weeks	Sequential	4
6.	Programming of core modules	Week 4	3 weeks	Sequential	4
7.	Programming of supporting modules	Week 4	3 weeks	Sequential	5
8.	Quality assurance of core modules	Week 5	1 week	Sequential	6
9.	Quality assurance of supporting modules	Week 5	1 week	Sequential	7
10.	Core module training	Week 7	1 day	Parallel	6
11.	Development of accounting reporting	Week 6	1 week	Parallel	5
12.	Development of management reporting	Week 6	1 week	Parallel	5
13.	Development of management analysis	Week 6	2 weeks	Sequential	5
14.	Detailed training	Week 7	1 week	Sequential	1-13
15.	Documentation	Week 4	2 weeks	Parallel	13

The start week shows when resources become available. Whether a task is parallel or sequential depends largely on context. All activities should be written in a list. For this example, we will use the activities

listed below in determining the critical path of getting ready for a day.

Table 5.3. Task to be Analyzed by CPM

NOTES

Task
Wake up
Shower
Get ready
Dress
Eat
Make bed
Review to do list
Brush teeth
Drive to work
Step 2 Analyze List

Step 2: Analyze List

Critical Path Analyses are presented using circle and arrow diagrams. In these, circles show events within the project, such as the start and finish of tasks. Circles are normally numbered to allow you to identify them.

An arrow running between two event circles shows the activity needed to complete that task. A description of the task is written underneath the arrow. The length of the task is shown above it. By convention, all arrows run left to right. An example of a very simple diagram is shown below:

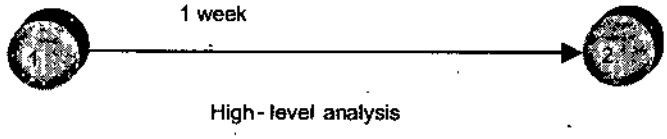

Fig. 5.7 Simple arrow and circle diagram

This shows the start event (circle 1), and the completion of the 'High Level Analysis' task (circle 2). The arrow between them shows the activity of carrying out the High Level Analysis. This activity should take 1 week. One activity cannot start until another has been completed; we start the arrow for the dependent activity at the completion event circle of the previous activity. An example of this is shown in Fig. 5.8 below:

NOTES

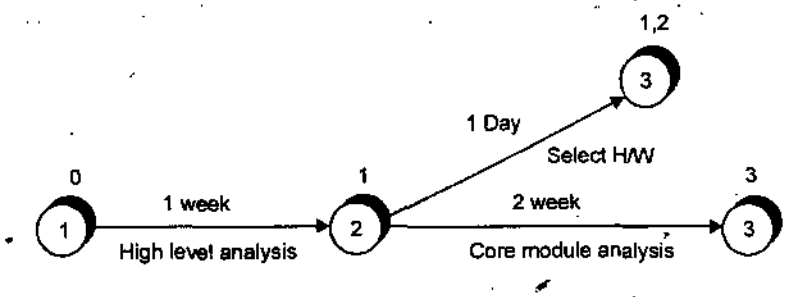

Fig. 5.8 Circles and arrow diagram showing two activities that cannot be started until the first activity has been completed

Here the activities of "Selecting Hardware' and 'Core Module Analysis' cannot be started until 'High Level Analysis' has been completed. This diagram also brings out a number of other important points:

- Within Critical Path Analysis, we refer to activities by the numbers in the circles at each end. For example, the task 'Core Module Analysis' would be called 'activity 2 to 3'. 'Select Hardware' would be 'activity 2 to 4'.
- Activities are not drawn to scale. In the diagram above, activities are 1 week long, 2 weeks long and 1 day long. Arrows in this case are all the same length.

In the example above, you can see numbers above the circles. These show the earliest possible time that this stage in the project will be reached. Here units are whole weeks. A different case is shown below:

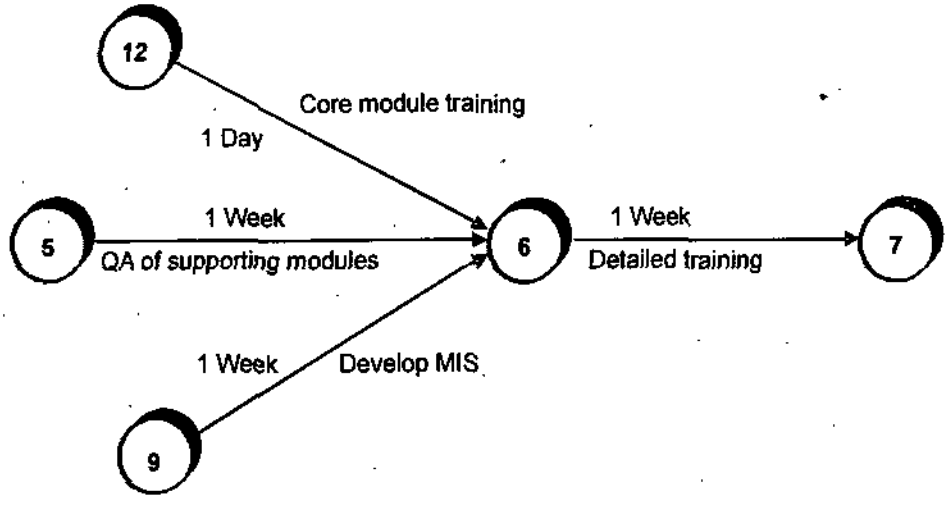

Fig. 5.9 Circle and arrow diagram showing an activity (6 to 7) that cannot start until other activities (12 to 6, 5 to 6, and 9 to 6) have been completed.

Here activity 6 to 7 cannot start until the other three activities (12 to 6, 5 to 6 and 9 to 6) have been completed. As an example, see fig. 5.10 for the full circle and arrow diagram for the computer project.

NOTES

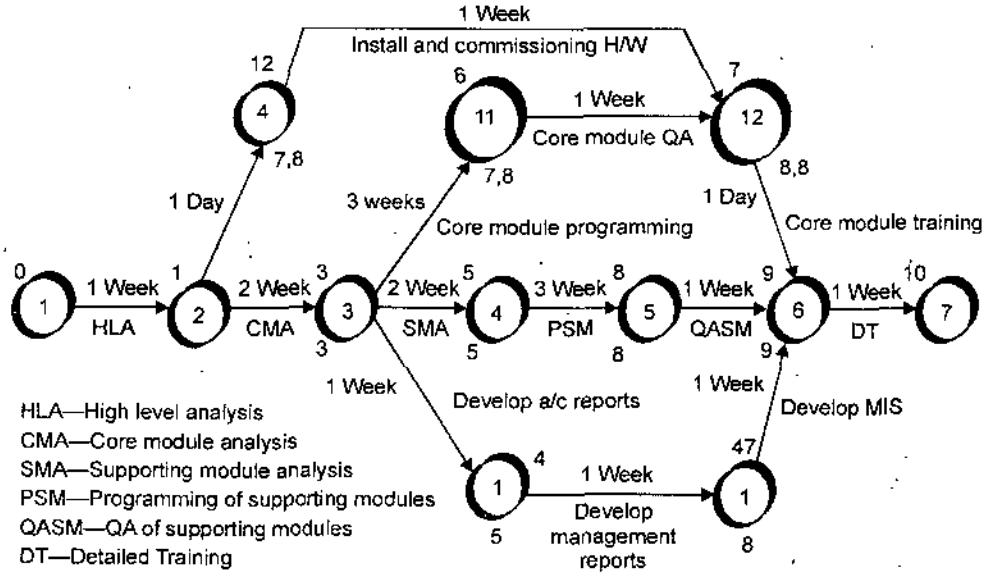

Fig. 5.10 Critical path analysis for a software project

This shows all the activities that will take place as part of the project. Notice that each event circle has a figure below it as well as a figure above. This shows the latest time that it can be reached with the project still being completed in the minimum time possible. You can calculate this by starting at the last event (in this case number 7), and working backwards.

You can see that event 4 can be completed any time between 1.2 weeks in and 7.8 weeks in. The timing of this event is not critical. Events 1 to 2, 2 to 3, 3 to 4, 4 to 5, 5 to 6 and 6 to 7 must be started and completed on time if the project is to be completed in 10 weeks. This is the 'critical path'—these activities must be very closely managed to ensure that activities are completed on time. If jobs on the critical path slip, immediate action should be taken to get the project back on schedule. Otherwise completion of the whole project will slip.

Crash Action

You may find that you need to complete a project earlier than your Critical Path Analysis says is possible. In this case you need to take action to reduce the length of time spent on project stages. You could mound resources into every project activity to bring down time spent on each. This would probably consume huge additional resources. A more efficient way of doing this would be to look only at activities on the critical path.

As an example, it may be necessary to complete the computer project in figure 5 in 8 weeks rather than 10 weeks. In this case you could look at using two analysts in steps $i2$ to $3i$ and $i3$ to $4i$, and two programmers instead of one in step $i4$ to $5i$. This would shorten the

project by two weeks, but would raise the project cost doubling resources at any stage often only improve productivity by, say, 50%. This occurs as time spent on coordinating the project consumes time gained by increasing resource.

As with Gantt Charts, in practice project managers tend to use software tools like Microsoft Project to create CPA Charts. Not only do this ease make them easier to draw, they also make modification of plans easier and provide facilities for monitoring progress against plans. Microsoft Project is reviewed at the top of our left hand title bar.

In this step, each process is ordered in the approximate order of completion. All processes are also given an estimated amount of time required to complete the task and all predecessor processes are also recorded. For example, the following table shows that in order to eat breakfast the person must have already dressed them and gotten ready for the day.

Table 5.4. Scheduled Tasks and their Completion Time

S. No.	*Task*	*Time (Minutes)*	*Predecessor*
1	Wake up	5	—
2	Shower	10	1
3	Get ready	20	2
4	Dress	10	2
5	Eat	15	3, 4
6	Make bed	5	1
7	Review to do list	10	5
8	Brush teeth	5	5
9	Drive to work	20	6, 7, 8

Step 3: Creating the Diagram

In this step we arrange the events on a CPM diagram showing the events with numbered boxes and the antecedent relationships with connection lines. There are many methods of constructing a CPM diagram, but we choose to use boxes instead of circles. Each area of the boxes is explained in the box below. Also, some diagrams prefer to include the time of the project on the relationship line, but we believe the method below is thorough and precise.

Example Box:

ID (ID #)	**ES** (Earliest Start Time)	**EF** (Earliest Finish Time)
ID (Task duration)	**LS** (Latest Start Time)	**LF** (Latest Finish Time)

NOTES

Step 4: Earliest Start Times

In this step we determine the earliest times for the starting and ending events. Event 1 can be started immediately, but events 2 and 6 cannot begin until event 1 is completed, 5 minutes later. Similarly, event 9 cannot begin until events 6, 7, and 8 are finished. Since event 9 must wait for all three of these activities to be completed, its finishing time will be the length of its task added to its starting time, which will be equal to the largest early finish time of the directly preceding events. The example of our CPM diagram with the earliest start times and earliest finish times can be seen below.

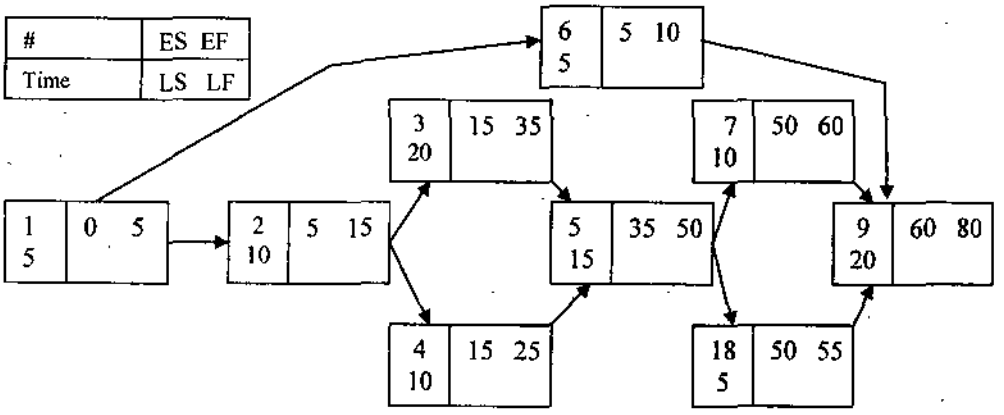

Fig. 5.11 Activities with ES and EF

Step 5: Latest Start Times

Determine the latest times for the ending events. This is completed by working backwards through the diagram. In each case, the latest starting time of an activity is equal to the earliest starting time of the activity directly following it minus the time required for the current project. In our example, as shown below, activity 5 has a latest starting time of 35 (50—15).

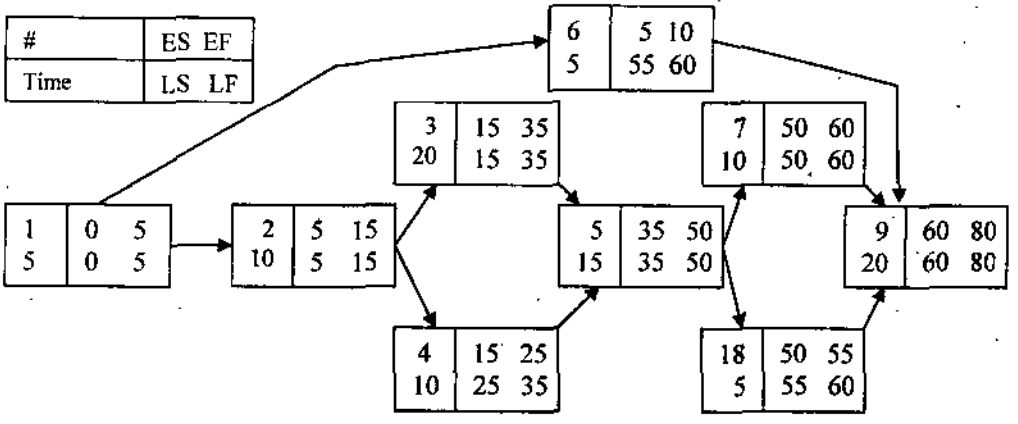

Fig. 5.12 Activities with ES, EF, LS, and LF

Step 6: Finding the Critical Path

If the earliest time that an event can be started is the same as the latest that it could be started (ES = LS), then the timing of that event is critical. The critical path is found by connecting the critical events, and is shown by double lines on the diagram. One must be aware that it is possible to have two separate critical paths. All other activities that are not part of the critical path have "slack" time, or time that an activity can be delayed without affecting the completion time of the project. Below is the completed diagram with the critical path.

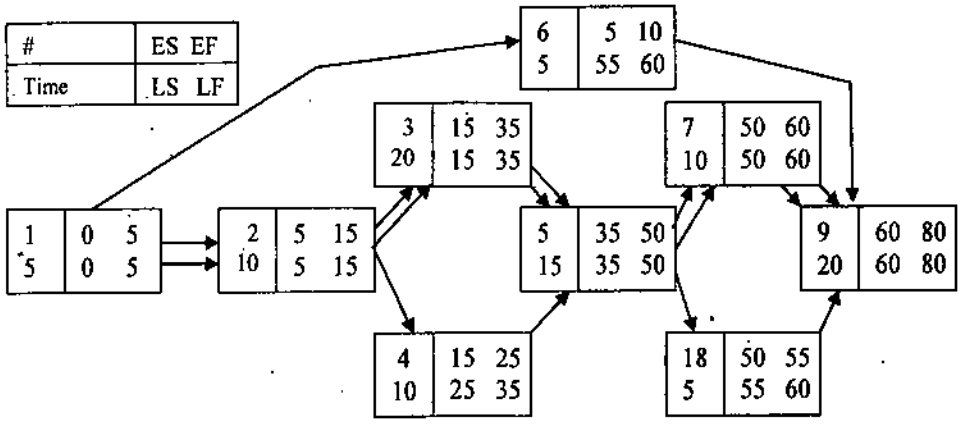

Fig. 5.13 Completed diagram with the critical path

Advantages of Using a CPM

- Helpful for scheduling, monitoring, and controlling projects.
- A project manager can determine actual dates for each activity and compare what should be happening to what is taking place and react accordingly.
- The activities and their outcomes can be shown as a network.

Displays dependencies to help scheduling.

- Evaluates which activities can run parallel to each other.
- Determines slack and float times.
- Widely used in industry.
- Can define multiple, equally critical paths.
- CPM determines the project duration, which minimized the sum of direct and indirect costs.

Disadvantages of Using a CPM

However, there are drawbacks of this technique, as estimations are used to calculate times; if one mistake is made the whole analysis could be flawed causing major upset in the organization of a project.

NOTES

- CPM's can be complicated, and complexity increases for larger projects.
- It does not handle the scheduling of personnel or the allocation of resources.
- The critical path is not always clear and needs to be calculated carefully.
- Estimating activity completion times can be difficult.

Determining the Critical Path

Adding the times for the activities in each sequence and determining the longest path in the project determine the critical path. The critical path determines the total calendar time required for the project. If activities outside the critical path speed up or slow down (within deadline) the total project time does not change. The amount of time that a non-critical path activity can be delayed without delaying the project is referred to as *slack time*.

If the critical path is not immediately obvious, it may be helpful to determine the following four quantities for each activity:

- ES—Earliest Start time
- EF—Earliest Finish time
- LS—Latest Start time
- LF—Latest Finish time.

These times are calculated using the expected time for the relevant activities. The earliest start and finish times of each activity are determined by working forward through the network and determining the earliest time at which an activity can start and finish considering its predecessor activities. The latest start and finish times are the latest times that an activity can start and finish without delaying the project. LS and LF are found by working backward through the network. The difference in the latest and earliest finish of each activity is that activity's slack. The critical path then is the path through the network in which none of the activities have slack.

Note: The slack of an event is a measure of the excess time and resources available in achieving this event. Positive slack would indicate *ahead of schedule*; negative slack would indicate *behind schedule*; and zero slack would indicate *on schedule*.

The variance in the project completion time can be calculated by summing the variances in the completion times of the activities in the critical path. Given this variance, one can calculate the probability that the project will be completed by a certain date assuming a normal probability distribution for the critical path. The normal distribution assumption holds if the number of activities in the path is large enough for the central limit theorem to be applied.

NOTES

Since the critical path determines the completion date of the project, adding the resources required to decrease the time for the activities in the critical path can accelerate the project. Such a shortening of the project sometimes is referred to as *project crashing.*

Update as Project Progresses

Make adjustments in the PERT chart as the project progresses. As the project unfolds, the estimated times can be replaced with actual times. In cases where there are delays, additional resources may be needed to stay on schedule and the PERT chart may be modified to reflect the new situation.

Why Critical Path Analysis and PERT Charts

Basically Critical Path Analysis and PERT Charts are used for planning and scheduling more complex projects. Critical Path Analysis and PERT are powerful tools that help you to schedule and manage complex projects. They were developed in the 1950s to control large defense projects, and have been used routinely since then.

As with Gantt Charts, Critical Path Analysis (CPA) helps you to plan all tasks that must be completed as part of a project. They act as the basis both for preparation of a schedule, and of resource planning. During management of a project, they allow you to monitor achievement of project goals. They help you to see where remedial action needs to be taken to get a project back on course.

The benefit of using CPA over Gantt Charts is that Critical Path Analysis formally identifies tasks which must be completed on time for the whole project to be completed on time, and also identifies which tasks can be delayed for a while if resource needs to be reallocated to catch up on missed tasks. The disadvantage of CPA is that the relation of tasks to time is not as immediately obvious as with Gantt Charts. This can make them more difficult to understand for someone who is not familiar with the technique.

A further benefit of Critical Path Analysis is that it helps you to identify the minimum length of time needed to complete a project. Where you need to run an accelerated project, it helps you to identify which project steps you should accelerate to complete the project within the available time. This helps you to minimize cost while still achieving your objective.

CPM Vs. PERT

Both CPM and PERT (Program Evaluation and Review Technique) provide the user with project management tools to plan, monitor, and

NOTES

update their project as it progresses. There are many similarities and differences between the two, however.

Similarities between PERT and CPM

- Both follow the same steps and use network diagrams.
- Both are used to plan the scheduling of individual activities that make up a project.
- They can be used to determine the earliest/latest start and finish times for each activity.

Differences between PERT and CPM

- PERT is probabilistic whereas CPM is deterministic.
- In CPM, estimates of activity duration are based on historical data.
- In PERT, estimates are uncertain and we talk of ranges of duration and the probability that an activity duration will fall into that range.
- CPM concentrates on Time/Cost trade off.

5.6 SIMULATION SOFTWARE

Simulation software is based on the process of imitating a real phenomenon with a set of mathematical formulas. It is, essentially, a program that allows the user to observe an operation through simulation without actually performing that operation. Simulation software is used widely to design equipment so that the final product will be as close to design specs as possible without expensive in process modification. Simulation software with real-time response is often used in gaming, but it also has important industrial applications. When the penalty for improper operation is costly, such as airplane pilots, nuclear power plant operators, or chemical plant operators, a mock up of the actual control panel is connected to a real-time simulation of the physical response, giving valuable training experience without fear of a disastrous outcome.

Advanced computer programs can simulate weather conditions, electronic circuits, chemical reactions, mechatronics, heat pumps, feedback control systems, atomic reactions, even biological processes. In theory, any phenomena that can be reduced to mathematical data and equations can be simulated on a computer. Simulation can be difficult because most natural phenomena are subject to an almost infinite number of influences. One of the tricks to developing useful simulations is to determine which are the most important factors that affect the goals of the simulation.

In addition to imitating processes to see how they behave under different conditions, simulations are also used to test new theories. After creating a theory of causal relationships, the theorist can codify the relationships in the form of a computer program. If the program then behaves in the same way as the real process, there is a good chance that the proposed relationships are correct.

5.7 SIMULATION LANGUAGES

We are finally ready to describe or program the model in a language acceptable to the computer to be used. Well over a hundred different simulation languages are commercially available. In addition there are literally hundreds of other locally developed languages in use in Companies and Universities. We have three generic choices, namely: Build the model in a general-purpose language, build the model in a general-purpose simulation language, use a special purpose simulation *packages.*

Although general purpose programming languages such as FORTRAN, C++, Visual Basic, or Pascal can be used they very seldom are anymore. Using one of the general or special purpose simulation packages has distinct advantages in terms of ease, efficiency and effectiveness of use. Some of the advantages of using a simulation package are reduction of the programming task, provision of conceptual guidance, increased flexibility when changing the model, fewer programming errors, automated gathering of statistics.

The goal of any simulation package is to close the gap between the user's conceptualization of the model and an executable form. Simulation packages divide themselves more or less into two categories, namely (*a*) general purpose simulation languages, and (*b*) special purpose simulators. In the first category are those which can solve almost any discrete simulation problem. Among these are such systems as ARENA, AweSim, GPSS, SimScript II.5, Extend etc. Some systems are used for the simulation of manufacturing and material handling problems. Packages such as SimFactory, ProModel, AutoMod, Taylor II, and Witness fall into this category. Others are designed for conducting Business Process Reengineering studies. These include BPSimulator, ProcessModel, SIMPROCESS, etc. Still others are for healthcare delivery (for example MedModel), or communications networks (for example COMNET II.5).

Early effort in a simulation study is concerned with defining the system to be modeled and describing it in terms of logic flow diagrams and functional relationships. But eventually one is faced with the problem of describing the model in a language acceptable to the computer to be used. Most digital computers operate in a binary method of data

NOTES

representation, or in some multiple of binary such as octal or hexadecimal. Since these are awkward languages for users to communicate with, programming languages have evolved to make, easier to converse with the computer. Unfortunately, so many general and special purpose programming languages have been developed over the years, that it is a nearly impossible task to decide which language best fits or is even a near best fit to any particular application.

Over 170 programming languages were in use in the United States in 1972 and today there are even more. Consequently, the usual procedure is to use a language known by the analyst, not because it is best, but because it is known. It should be stated that any general algorithmic language is capable of expressing the desired model; however, one of the specialized simulation languages may have very distinct advantages in terms of ease, efficiency and effectiveness of use. It is not the purpose of this chapter to teach how to program in any of the languages described, nor to discuss implementation techniques. What we do hope to accomplish is to make the reader aware of the characteristics of some of the more popular languages, their strengths and weaknesses. The major differences between special purpose simulation languages in general are classified as : (1) the organization of time and activities, (2) the naming and structuring of entities within the model, (3) the testing of activities and conditions between elements, (4) the types of statistical tests possible on the data and (5) the ease of changing model structure.

A computer simulation language describes the operation of a simulation on a computer. There are two major types of simulation: continuous and discrete-event simulation languages. Though there are several modern languages that can handle combinations of continuous and discrete-event simulation languages. Most languages also have a graphical interface and at least simple statistical gathering capability for the analysis of the results. An important part of discrete-event languages is the ability to generate pseudo-random numbers and variates from different probability distributions.

5.8 CONTINUOUS SYSTEM SIMULATION LANGUAGES

Continuous simulation languages developed in late fifties as simulators of analog computers. Simulation of analog computers is based on creating an analog electronic system whose behavior is described by the same mathematical model (with the help of differential equations) as the system being investigated. Interconnecting standard blocks based mostly on operational amplifiers modified to act as integrators, adders, and other functional units creates the electronic system. Then the user

performs experiments with this electronic system by applying suitable inputs and recording the voltage at certain output points (for example, oscilloscope, and plotter). The changing voltage represents a time function that is the same as the function that describes the changes in the original system whose physical nature may be totally different (for example, mechanical displacement, temperature, etc.). The main problem of analog computers is an analog implementation of certain operations like multiplication, generation of some functions, generation of delays and others.

Digital computers perform all these functions very easily and today continuous simulation is performed only to them. Nevertheless there is one operation where the analog computers are better, is integration. Digital computers use numerical integration that is generally slower and less accurate compared with the integration of an analog integrator. Some special applications based on fast response use therefore the so-called Hybrid computers, which contain analog and digital parts connected by analog to digital and digital to analog converters. The digital part does everything except integration. It computes inputs of integrators, which are then converted by digital to analog converters to analog signals inputted to analog integrators. Their outputs are treated in the opposite way. The digital part also controls the interconnection of the analog part, which might thus change during computation.

Continuous simulation languages can be viewed as the model essentially as a set of differential equations. We have a specific kind of continuous simulation language named Advanced Continuous Simulation Language (ACSL), which supports textual or graphical model specification.

Advanced Continuous Simulation Language (ACSL)

The Advanced Continuous Simulation Language, or ACSL (pronounced as axle), is a computer language designed for modeling and evaluating the performance of continuous systems described by time-dependent, nonlinear differential equations. It is a tongue of the Continuous System Simulation Language (CSSL), originally designed by the Simulations Council Inc (SCI) in 1967 in an attempt to unify the continuous simulations field.

ACSL is an equation-oriented simulation language consisting of a set of arithmetic operators, standard functions, a set of special ACSL statements, and a MACRO capability which allows extension of the special ACSL statements. ACSL is intended to provide a simple method of representing mathematical models on a digital computer. Working from an equation description of the problem or a block diagram, the user writes ACSL statements to describe the system under investigation. An important feature of ACSL is its sorting of the continuous model equations, in

NOTES

contrast to general purpose programming languages such as Fortran where program execution depends critically on statement order.

Typical Applications

Applications of ACSL in new areas are being developed constantly. Typical areas in which ACSL is currently applied include control system design, aerospace simulation, chemical process dynamics, power plant dynamics, plant and animal growth modeling, toxicology models, vehicle handling, microprocessor controllers, and robotics.

Classification of Continuous Simulation Languages

Block oriented simulation languages: Block oriented simulation languages are based on the methodology of analog computers. The system must be expressed as a block diagram that defines the interconnection of functional units and their quantitative parameters. "Programming" means entering the interconnection of the blocks and their description. Then the user adds statements and/or directives that control the simulation. If the system is described as a set of equations, they must be converted to a block diagram. This conversion is a simple straightforward process. The typical blocks available in most continuous block oriented languages are integrators, limiters, delays, multipliers, constant values, adders, holders, gain (coefficient) and other. For example GPSS is the Block oriented simulation languages.

Expression oriented continuous languages: Expression oriented continuous languages are based on writing expressions (equations) that represent the mathematical model. So the system simulated must be expressed by a set of equations. Then the user adds statements and/or directives that control the simulation. Some languages enable both block and expression based ways of system definition. Simulation control means selection of: the integration method (because some languages offer more), the integration step, the variables (outputs of blocks) that should be observed, the intervals for collecting data for printing and/or plotting, scaling of outputs (that may be also done automatically), duration of the simulation runs, number of repetitions and the way certain values are changed in them, etc. SimScript is the best example of expression oriented simulation languages.

5.9 DISCRETE SYSTEM SIMULATION LANGUAGES

Discrete-system simulation languages can be viewed as the model as a sequence of random events each causing a change in state of the system being simulated.

Classification of Discrete Simulation Languages

Broadly there are four types of discrete simulation languages.

Flowchart oriented languages: These are represented by the language GPSS (General Purpose Simulation System), which exists in many versions on various computers. The user must view the dynamics of the system as a flow of the so called transactions through a block diagram. Transactions are generated, follow a path through a network of blocks, and are destroyed on exit. In blocks transactions may be delayed, processed, and passed to other blocks. Blocks are in the program represented by statements that perform the activities of the model.

Activity oriented languages: These are not based on explicit scheduling of future activities of the system. For each activity the user describes the condition under which the activity can take place (that also covers scheduling if the condition is reaching certain time). The algorithm of the simulation control repeatedly increases time and tests conditions of all activities. The disadvantage of this approach is obvious, it is necessary to evaluate all conditions in every step, which may be very time-consuming. On the other hand it is conceptually very simple and the algorithm can be easily implemented in general high level languages (there are simulation languages based on this approach, but not widely used).

Event oriented languages: These are based on direct scheduling and cancelling of future events of the system. The approach is very general. The user must view the dynamics of the system simulated as a sequence of relatively independent events. Every event may schedule and/or cancel other events. The system routine must keep record of scheduled events. That is why every event is represented by the so called Event notice that contains the time, the event type, and other user data. Event notices are kept in the so called Calendar, where the event notices are ordered by the scheduled time. After completion of an *Event routine*, the system removes the event notice with the lowest time from the calendar, updates the model time by its time, and starts the corresponding routine. This is repeated until the calendar becomes empty or the program stops because of other reason. Scheduling means inserting event notices to the calendar by the scheduled time, canceling removes them. The approach based on explicit expressing of events is called Discrete Event Simulation. A typical representative of this group of languages is the language SIMSCRIPT, but its version II.5 supports also process oriented simulation.

Process oriented languages: These are based on the fact, that events are not independent. An event is typically a consequence of other previous events. In other words it is often possible to define sequences of events that may be viewed as entities of a simulation model at

NOTES

higher level of hierarchy. A sequence of events is called *Process*. Unlike events process has a dimension in time. Process based abstract systems are very close to reality, which is always made of various objects that exist and act in parallel interfering with each other. Process way of viewing system dynamics is thus very natural. Mostly a process models an activity of a real object. It is believed, that process oriented discrete simulation is the best way how to create discrete simulation models. Typical representatives of this group of languages are MODSIM, SIMSCRIPT II.5, and the system class simulation of the SIMULA language.

5.10 GPSS (GENERAL PURPOSE SIMULATION SYSTEM)

The first version of the GPSS language was introduced by IBM in October, 1961. Thus, it has been in continuous, world-wide used for over 40 years. There are very few programming languages of any kind for which this is true. GPSS introduced the transaction-flow modeling paradigm. Under this paradigm, active "consumer" objects, called transactions, travel through a block diagram representing a system, competing for the use of "passive server" objects. This modeling paradigm is extremely general and has been adopted in a large number of simulation languages. Terminology differs from language to language. Transactions may be called items, entities, tokens, etc., but the basic architecture is the same: active objects compete for passive resources while travelling through a diagram representing a system.

GPSS was designed by Geoffrey Gordon as a tool for use by non-programmers. It is interesting to note that over 40 years later, "no programming required" is still featured as a virtue in advertisements for simulation software. Historically, GPSS models were developed by drawing block diagrams and manually typing character-based representations of blocks. Over time, the sizes of GPSS models grew well beyond what Geoffrey Gordon had ever anticipated. While drawing a 100-block diagram is relatively easy, drawing a 5000-block diagram is difficult. As users began developing larger and larger models, and as they became more comfortable with typing in blocks without first having drawn them, GPSS came to be viewed as a true programming language.

GPSS/H

GPSS/H was first placed into commercial service in November, 1977. Over the ensuing years, it has become well known for its speed and dependability. In GPSS/H, the transaction flow world view has been extended with many advanced features to make an extremely powerful and flexible tool capable of handling large, complicated models with

NOTES

ease, yet still providing exceptionally high performance. GPSS/H has been a very successful product. Its success is the result of both the superiority of its original design and subsequent years of improvement and enhancement. Most of the enhancements developed for GPSS/H have been improvements to it as a programming language.

Since it is a simulation language, GPSS/H requires some programming-style effort, but it does so within the intuitive modeling framework Gordon designed to be readily used without extensive programming experience. It is well suited for modeling both simple systems and large, complex systems. Although many new simulation tools have been introduced over the past decade, they have often been designed for specific classes of applications. In strong contrast, GPSS/H continues to be one of the most general, flexible, and *powerful* simulation environments currently available. GPSS/H is in use around the globe modeling manufacturing, transportation, distribution, telecommunications, hospitals, computers, logistics, mining, and many other types of queueing systems.

Representing a System in GPSS/H. In a GPSS/H model a transaction can represent a patient, a telephone call, a truck, a data packet, or any other type of individually identifiable entity that has a *behavior*. As a model executes, many transactions can flow through the model simultaneously—just as many "objects" would move through the real-world system being modeled. In addition, while flowing through the model, multiple transactions can execute the same GPSS/H model statements at the same instant in simulated time without any intervention by the modeler. The execution of a process-interaction simulation model is thus similar to a multi-threaded computer program. This differs greatly from the single-threaded, sequential execution of most general-purpose programming languages, and is a good reason why such languages are usually poor tools for writing simulation models.

Many simulation projects focus on the optimal use of system resources such as people, machines, conveyors, computers, physical space, and so on. In a GPSS/H simulation model, transactions compete for the use of these system resources. As transactions flow through the process representation, they automatically queue up when they cannot gain control of a necessary resource. The modeler does not have to specify the transaction's waiting time or its queueing behavior for this to occur. Hence, the passage of time in a GPSS/H model can be represented *implicitly*, as in the case of a part waiting for a machine to become free, or *explicitly*, as in the case of a part being processed by a machine.

A GPSS/H model, like most real-world systems, may consist of multiple processes operating at the same time. Furthermore, each such process may affect the other processes in the system. For example, two parallel manufacturing processes may converge to a single inspection point

NOTES

where they are competing for a single resource, *e.g.*, an inspector. GPSS/H provides the capability for multiple parallel processes to interact with each other automatically. Transactions ("objects") may be sent between processes; they may control or share common resources.

Ease-of-Use. The presence or absence of a single characteristic such as a "visual" interface for building models is insufficient to determine a tool's ease-of-use. Whether a tool is "easy to use" depends upon the combination of *general characteristics* and *specific features* that are heavily used when real-world models are built. Moreover, the ease-of-use that is claimed for almost all simulation tools can mean many things—that the tool is easy to learn, easy to use once learned, easy to use when modifying models, easy to use when building simple models, or easy to use when building large, complex models. Rarely is a product equally good at all of these, but equally rarely is the prospective user informed about which "ease-of-use" is being claimed.

A simulation language's ease-of-use is rarely constant over the duration of a simulation project. The ease-of-use of a tool that is optimized for developing small models may diminish as a model gets larger and larger, and more details of system operation have to be incorporated into a model. Conversely, a tool that requires a bit more study to master may be harder to use in the early stages of a project, but offer downstream rewards when it is able to meet modeling requirements for a large, complex system. Scalability of technology is an important contributor to ease-of-use. GPSS/H has excellent scalability. It is able to handle very large models (10's of thousands of blocks).

Model Validation and Debugging. The GPSS/H Interactive Debugger provides invaluable assistance for rapid model development and verification. Prior to the advent of GPSS/H in 1977, most simulation tools featured batch-oriented, non-interactive debugging. To debug a model, one typically generated a large trace file and very carefully proofread the trace file to verify model correctness.

In GPSS/H, simple debugger commands are used to control a model's execution and to examine its status. Functions are provided to "step" through the model, to set breakpoints and traps that interrupt model execution based on multiple criteria, and to return to a previously saved state of the model. Almost all data values can be examined, including local data, global data, transaction attributes, entity statistics, and array data values. The debugger provides a "windowing" mode that displays source code, model status, and interactive user input as the model runs. A modeler can interrupt a long-running model at any time and use the debugging features to make sure that everything is running correctly before resuming execution. The GPSS/H debugger has almost no effect on execution speed. Because of this, many modelers use the debugger as their everyday run-time environment for GPSS/H.

General Purpose Simulation System is a discrete-time simulation language, where a simulation clock advanced in uniform discrete steps. A system was modeled as transactions entered the system and were passed from one service to another. This was particularly well suitable for problems such as a factory. It was popular in the late 1960s and early 1970s but is little used today. GPSS is less flexible than simulation languages such as SIMULA and SIMSCRIPT II.5 that model a system with discrete events that are not constrained by a uniform time step.

General Purpose Simulation System is a language designed for discrete event modeling. The language as it exists today is the result of more than twenty years of evolution. While GPSS has its roots in the early days of mainframe computing, its basic ideas have proven suitable for application to today's problems using modern computing environments. The popularity of GPSS is due, in part, to its power of expression. A short, easily understood GPSS model would require many pages of FORTRAN coding to accomplish a similar goal. The GPSS user is free to concentrate on the important issues in the model being developed since the language itself collects statistics, produces tabulated results and performs a host of mundane tasks one would prefer not to deal with. The purpose of this brief article is to present some basic terminology and concepts which will enable the person meeting GPSS for the first time to understand the broad range of applications, power and ease of use that this language brings to the simulationist.

GPSS provides a set of abstract components of various types and a set of operators called blocks which perform certain actions on the individual components. The transaction is the component which moves through a sequence of blocks that has been designed to model the system being studied. The state of the components of the model determines the details of how a block of a given type will operate. For example, a block which permits a transaction to take control of a piece of equipment will not allow the transaction to proceed if the equipment is already at maximum capacity.

Several different types of equipment components are available in GPSS. A facility is an entity which is either available for use or is in use by at most one transaction at a time. Storage is similar but has a capacity which may be specified to suit the needs of the model builder. Finally, a logic switch is a simple ON/OFF element which may be set and tested to modify the path of a transaction through the blocks of the model.

GPSS provides an underlying control mechanism which is transparent to the user but which ensures that competition between transactions (for use of a facility, for instance) is consistently arbitrated and that transactions are moved through the blocks of the model in an orderly and efficient way. This control mechanism also administers the GPSS clock which provides the time-base for models.

NOTES

The GPSS clock is also an abstraction. It measures time in clock units. A clock unit is interpreted appropriately by the person designing the model. In the factory model a clock unit could correspond to a second or a minute, while in the communication network the finer resolution of a one-millisecond clock unit would be more suitable Function entities are provided to describe various types of numeric relationships. They may be used to produce random variates from theoretical or empirical probability distributions. They may also be used to modify transaction behavior depending on system-wide, entity or transaction attributes. Variable entities can be used to perform arithmetic calculations in all implementations of GPSS. The greater flexibility of GPSS/VX and GPSS/C versions permits arithmetic expressions to be entered wherever a constant is permitted. Certain modeling situations may take advantage of the set concepts embodied in the group entity-type. Other situations will benefit from the use of COUNT and SELECT blocks which permit a number of entities to be examined in a single operation.

GPSS provides entities known as save-values and matrix save-values which are numeric storage locations available for use or reference by all transactions. In addition, each transaction has a set of private numeric attributes called parameters which may be used to hold a variety of information as required by the user, such as a part number code or the size or weight of an item represented by a transaction.

A rich set of Standard Numerical Attributes (SNA's) provide a convenient notational way to access components of a model. For instance, Q$BOXOFFICE represents the current contents of the queue entity named BOXOFFICE. This SNA may be used wherever a numeric value is allowed. A simple balking situation could be represented by the block:

TEST L Q$BOXOFFICE,FN$BALK,GOAWAY

which allows transactions to pass through if the current length of the BOXOFFICE lineup is less than the value returned by the function BALK, and otherwise re-directs them to the block GOAWAY. BALK may be written to return a random variate from any desired probability distribution; or it may return a value varying with the time of day in the model or a value dependent on the number of packages (represented by a transaction parameter) the theatre patron is carrying.

Simulation Software's GPSS products all feature interactive debugging capability which allows setting breakpoints in the model, single-stepping and examining attributes of model entities. Each implementation has additional innovative tools designed to make debugging a much shorter task. Features such as conditional breakpoints and animated displays optimize the model-development process, while internally-compiled

code and optimizations shorten the run-times for production model runs.

A Simple GPSS Model

NOTES

The following brief example models a bank with a individual queues for each of five tellers. The transactions are customers and facilities are used to represent the bank tellers.

```
1.  SIMULATE
2.  LINES EQU 1(5),Q,F
3.  ARRIVE FUNCTION RN5,BE
4.  GENERATE 20,FN$ARRIVE
5.  SELECT MIN 1,LINES,LINES+4,Q
6.  QUEUE P1
7.  SEIZE P1
8.  DEPART P1
9.  ADVANCE 90,20
10. RELEASE P1
11. TERMINATE 1
12. START 100
13. END
```

The following is a line-by-line description of the model:

1. SIMULATE is a control statement.
2. EQU defines LINES as the name of the first of a group of 5 corresponding queues and facilities—the lineups and tellers.
3. Defines ARRIVE as the name of a function transforming a random number between 0 and 1, into a value from a built-in exponential (BE) distribution–a commonly-used in simulating arrival processes. The random number is taken from the fifth built-in stream of such numbers (RN5).
4. The customers arrive an average of 20 seconds apart with a random variation determined by sampling the function ARRIVE is defined in the previous line.
5. Set parameter 1 of each entering transaction to the number of the queue with the MINimum length.
6. Join the queue whose number is now in parameter 1.
7. When the corresponding facility is available take possession of it.

NOTES

8. Now leave the lineup, recording the delay time in the queue statistics.
9. Delay for an average of 90 seconds with a variation of 20 seconds either way, representing time for the banking to be done. The delay is chosen randomly with uniform probability in the 70 to 110 second range. The facility remains in use for this period. The random number used to determine the actual delay time is taken from the first random number stream, thus the arrival times and delay times are statistically independent.
10. Free the facility for use by the next transaction.
11. Destroy the transaction. Count 1 completes customer cycle.
12. Run the model until 100 customers have reached the TERMINATE block.
13. END marks the last line of the model.

Syntax in GPSS

Block Entities. The GPSS block entity is the basic structural element of a GPSS simulation program. It is useful to think about a GPSS model by considering its Block diagram as implemented in a simulation program. The model is the connected network of block symbols that correspond to the positions of block entities (lines of GPSS code) in the simulation. The types of Blocks are identified by names, usually written in upper case, that suggest their function, such as GENERATE, SEIZE, and TERMINATE. Each type of block is associated with an action in the simulation. All Blocks have a set of operands that are written sequentially after the Block type and separated by commas. The sequential positions of these operands are referred to by capital letters (A,B,C,...) of the alphabet in their usual order. Blocks can optionally be given unique labels suitable to the model they represent. The syntax of a Block statement is then:

Label BLOCKTYPE A,B,C,...;comment

Operands may be blank, in which case a default value is used. If all trailing Operands are blank they may be omitted altogether.

System Numerical Attributes. System Numerical Attributes, or SNAs, are simulation "state variables" that are available for use throughout a simulation. They return numeric or string values, and may be used in GPSS Statement operands and in Expressions. They are best considered with the entities they describe.

Transactions. A Transaction is a GPSS object with a set of attributes. Attributes are called **Parameters** in GPSS. Transactions are created either one at a time or in batches in GENERATE Blocks and pass from Block to Block in a simulation acting on, and being acted upon

by, the Blocks they pass through, and possibly by other GPSS Entities. A single Transaction may consist of several individual entities, such as more than one person being "served" by an elevator. Common ways to describe how Transactions behave in a GPSS simulation is to use phrases such as "Transactions enter, pass through, are contained in, acquire, own, and/or are in contention for various Blocks." Also Block types, which are usually verbs, can have case suffixes as in "a *Transaction SEIZEs a Facility*", *where SEIZE is a Block type.* This terminology is patterned after the idea that an entity, represented as a Transaction, actively deals with various types of servers, as in a single server queueing system. Each Transaction in the model is contained in exactly one Block, but most Blocks may contain many Transactions. Each Transaction enters one Block, then the next, and so on, until it is TERMINATEd or the simulation ends. Transactions occasionally must wait in or before a Block until conditions are favorable for entry into the that Block. The Blocks a transaction passes through are written in successive lines of a GPSS simulation unless they encounter a TRANSFER Block which may send them to another Block not in the sequence. A GENERATE Block creates Transactions for future entry into the simulation; it has 5 operands *(A,B,C,D,E)*:

A = Mean inter generation time or Function representing the distribution of inter generation times

B = Inter generation time half range (for uniformly distributed inter generation times) or Function Modifier.

C = Start delay time.

D = Maximum number of transaction to be created at this GENERATE Block.

E = Priority level.

It helps to think that the first transaction created by a GENERATE Block is actually created right before the GENERATE Block and then *enters the GENERATE Block.* From then on, every time a Transaction enters a GENERATE Block, the next Transaction is created. This new, next Transaction does not enter the GENERATE Block right away. Instead, it is placed on the Future Events Chain according to the arrival time interval specified by the GENERATE Block. The Future Event Chain (FEC) is a list of events to be executed later than the current simulation clock time. They are removed from the FEC in time sequence. The Current Events Chain (CEC) is the set of Transactions that still have Blocks to be entered at the current system clock time. Transactions are taken from the front of the Current Events Chain, one at a time, to enter as many Blocks as possible. When there are no more Transactions on the Current Events Chain, GPSS World will advance the system clock. ASSIGN Blocks are used

NOTES

to place or modify a value in a Transaction Parameter; if the Parameter does not exist it is created. Their syntax is:

ASSIGN A, B, C

Where,

A = Parameter number of the Active Transaction. It is not optional. Plus/minus sign after the number indicates that the value of the Operand B is to be added or subtracted to the original Parameter value.

B = the operand must be Name, Number, String, Parenthesized-expression, SNA, or SNA*Parameter.

C = Function number. It is optional.

Related SNA

P *Parameter* or **Parameter* (where *Parameter* is the number or the name of a Transaction Parameter): Parameter value. P *Parameter* returns the value of Parameter *Parameter*. (note: *e.g.*, P1 or *1 or P$NAME will yield the value in the Parameter 1 in the first two cases and the Parameter called NAME in the final case.). An important Parameter of each Transaction is its Mark Time, the time the transaction entered the simulation or when it passes through a MARK Block with no A Operand. The time from the last Mark Time to the current clock time can be retrieved from the M1 SNA. The current simulation time can be assigned to a transaction by a MARK Block. Its syntax is

MARK A

Where A =Parameter number. It is optional. If A is absent the current time is kept in a virtual Parameter called the Mark Time.

Facilities

A *Facility* is an entity that has several attributes, the most important of which is ownership. A Facility is automatically created when the first transaction reached a block that uses a Facility. Facilities do not have to be defined / declared by a Command.

There are several GPSS Blocks which can be used with Facilities:

- SEIZE Blocks attempt to take ownership of a Facility.
- RELEASE Blocks relinquish ownership of a Facility.
- PREEMPT Blocks attempt to take ownership of a Facility, possibly displacing the existing owner.
- RETURN Blocks relinquish ownership of a Facility.
- FAVAIL Blocks place a Facility in the available state.
- FUNAVAIL Blocks place a Facility in the unavailable state.

NOTES

Except for PREEMPT and FUNAVAIL, these Blocks all take a single Operand A, which is the label of the Facility to be SEIZEDed, RELEASEed, RETURNed, and/or made available. FUNAVAIL has up to 8 Operands; it is not used frequently enough to be considered in this document. The PREEMPT Block has 5 Operands:

A = Facility name or number.

B = PR for Priority Mode or, if omitted Interrupt Mode.

C = Block name or number for the new destination of the Transaction presently owning the Facility.

D = Parameter number of the preempted Transaction to receive residual time if preempted Transaction is removed from FEC.

E = When RE the preemption is in the Remove Mode, which Removes preempted Transaction from contention for the Facility. If RE, you must specify a destination in Operand C.

Transactions acquire ownership of a Facility by successfully entering a SEIZE or PREEMPT Block (a PREEMPT Block has the power to displace the existing owner of the Facility). A Facility is restricted to be owned by a single Transaction, in which case it is said to be busy, or it may not be owned at all, in which case it is said to be idle. If a Transaction cannot acquire ownership, it comes to rest on a Facility Transaction Chain. A Facility has several waiting lines for Transactions that are waiting for some Facility-oriented event to occur. Each Facility has a Delay Chain for normally waiting Transactions, a Pending Chain for Interrupt Mode preemptions that were not allowed, and an Interrupt Chain for previously preempted Transactions. Transactions waiting on a Delay Chain, a Pending Chain, or an Interrupt Chain are said to be "in contention", for the Facility. Each Transaction which owns a Facility must eventually give up ownership by entering a RELEASE or a RETURN Block. Since a contending Transaction will generally become the owner of the Facility, contention for a Facility carries the obligation of eventually releasing the Facility. Those Transactions which fail in their attempt to enter a SEIZE Block come to rest in priority order on the Delay Chain of the Facility. When a Facility is freed by an owning Transaction, the next owner is chosen from occupants of the Facility's Transaction chains. The pending Interrupt Mode preemptors are chosen first, followed by previously preempted Transactions, followed by Transactions waiting normally in priority order on the Delay Chain.

A Facility may be available or unavailable. When it is available, Transactions acquire and give up ownership of the Facility normally. When the Facility is unavailable, ownership is not given to newly arriving Transactions. FAVAIL Blocks are used to place a Facility in the available state, and FUNAVAIL Blocks are used to place it in the unavailable

NOTES

state. Any Transactions present at the Facility at the time it was made unavailable are disposed of as specified in the FUNAVAIL Block operands. The symbol Entnum stands for a Facility number of name. The SNAs associated with Facilities are:

- F *Entnum*—Facility busy. If Facility *Entnum* is currently busy, F *Entnum* returns 1. Otherwise F *Entnum* returns 0.
- FC *Entnum*—Facility capture count. The number of times Facility *Entnum* has been SEIZEd or PREEMPTed by a Transaction.
- FI *Entnum*—Facility *Entnum* interrupted. If Facility *Entnum* is currently preempted by a Transaction in an Interrupt Mode PREEMPT Block, FI*Entnum* returns 1. Otherwise FI*Entnum* returns 0.
- FR *Entnum*—Facility utilization. The fraction of time Facility *Entnum* has been busy. FR*Entnum* is expressed in parts-per-thousand and therefore returns a real value between 0 and 1000.
- FT *Entnum*—Average facility holding time. The average time Facility *Entnum* is owned by a capturing Transaction.
- FV *Entnum*—Facility in available state. FV*Entnum* returns 1 if Facility *Entnum* is in the available state, 0 otherwise.

Storage

A Storage Entity is associated with a number of storage units which are allocated and returned by Transactions. Storage entities must be by defining/declaring them with a STORAGE Command.

NAME STORAGE A

Name is a required label for this entity. A is required and is total storage capacity; this is often interpreted as the number of units of the Storage that are available to Transactions. There are several GPSS Blocks that can be used with Storage Entities:

- ENTER Blocks attempt to increase the contents of (place tokens in) a Storage Entity.
- LEAVE Blocks decrease the contents of (remove tokens from) a Storage Entity.
- SAVAIL Blocks place a Storage Entity in the available state.
- SUNAVAIL Blocks place a Storage Entity in the unavailable state.

The ENTER and LEAVE Blocks take two Operands, the first is required and indicates the Storage to be ENTERed or LEAVEd while the second is the number of units of the Storage capacity to be used or discarded

by the Transaction. If only one unit of Storage capacity is to be used or discarded, the second Operand can be omitted. SAVAIL and SUNAVAIL need only the A Operand which is the Storage to be made available or not. When a Transaction ENTERs a Storage Entity, it utilizes or occupies one or more storage units (given by the B Operand) of the Storage Entity. A Transaction is denied entry into an ENTER Block if its storage demand cannot be met. Such a Transaction comes to rest on the Delay Chain of the Storage Entity. Then it must wait until other Transactions free enough storage by entering LEAVE Blocks.

Storage capacity may be released by any Transaction, even if it had not previously ENTERed the Storage Entity. However, if more capacity is released than was declared in the STORAGE Command an Error Stop occurs.

When a Transaction enters a LEAVE Block and gives up one or more storage units, other Transactions are sought which can have their storage demands satisfied. A "first-fit-with-skip" discipline is used to schedule Transactions which are waiting. This means that each Transaction on the Delay Chain is tested for fit in the Storage Entity, starting with the highest priority. If a fit is found, the Transaction is removed from the Storage Delay Chain, allowed to enter the ENTER Block, and placed on the CEC behind its priority peers. Then the next Transaction on the Storage Delay Chain is tested. ENTER and LEAVE Blocks are used to update the statistics associated with a Storage Entity and several SNAs are available which return derived statistics. The SNAs associated with Storage Entities are:

- R*Entnum*—Unused storage capacity. The storage content (or spaces available for use by "tokens") available for use by entering Transactions at Storage Entity *Entnum*.
- S*Entnum*—Storage in use. S*Entnum* returns the amount of storage content (or "token" spaces) currently in use by entering Transactions at Storage *Entity Entnum*.
- SA*Entnum*—Average storage in use. SA*Entnum* returns the time weighted average of storage capacity (or "token" spaces) in use at Storage *Entity Entnum*.
- SC*Entnum*—Storage use count. Total number of storage units that have been acquired from Storage *Entity Entnum*.
- SE*Entnum*—Storage empty. SE*Entnum* returns 1 if Storage *Entity Entnum* is completely unused, 0 otherwise.
- SF*Entnum*—Storage full. SF*Entnum* returns 1 if Storage *Entity Entnum* is completely used, 0 otherwise.
- SR*Entnum*— Storage utilization. The fraction of total usage represented by the average storage in use at Storage Entity Entnum. SR*Entnum* is expressed in parts-per-thousand and

NOTES

therefore returns a real value between 0 and 1000, inclusively.

- *SMEntnum*—Maximum storage in use at Storage *Entity Entnum*. The "high water mark".
- ST*Entnum*—Average holding time per unit at Storage *Entity Entnum*.
- SV*Entnum*—Storage Entity in available state. SV*Entnum* returns 1 if Storage *Entity Entnum* is in the available state, 0 otherwise.

Queue Entities and Queue Blocks

Queue Entities are used primarily for the collection of statistics such as the current count, total entries, and the count-time product of a *Queue Block*. They keep track of transactions that are blocked from entering Facilities and Storages because the Storage is full. Similar to Facilities, Queues are created when the first entity reaches a block that used the Queue. The GPSS Blocks that can be used with Queue Entities are:

- QUEUE Blocks increase the content of a Queue Entity.
- DEPART Blocks reduce the content of a Queue Entity.

These Blocks have two Operands: A is the name or number of the Queue and B is the number of entities being QUEUEd or DEPARTing the Queue.

The usual procedure is to "sandwich" a SEIZE, PREEMPT, or ENTER Block between QUEUE and DEPART Blocks. Notice that the transaction does not DEPART from the all jobs Queue until after it has successfully SEIZEd the Maintenance repairman facility. The SNAs associated with Queue Entities are:

- Q*Entnum*—Current Queue content. The current count value of Queue *Entnum*.
- QA*Entnum*—Average Queue content. The time weighted average count for Queue *Entnum*.
- QC*Entnum*—Total queue entries. The sum of all Queue entry counts for Queue *Entnum*.
- QM*Entnum*—Maximum Queue contents. The maximum count of Queue *Entnum*.
- QT*Entnum*—Average Queue residence time. The time weighted average of the count for Queue *Entnum*.
- QX*Entnum*—Average Queue residence time excluding zero entries. The time weighted average of the count for Queue *Entnum* not counting entries with a zero residence time.
- QZ*Entnum*—Queue zero entry count. The number of entries of Queue *Entnum* with a zero residence time.

Frequency distributions of these statistics can be accumulated by the use of QTABLE Commands that initialize queue time frequency distribution tables. The syntax of a QTABLE command is:

NAME QTABLE A,B,C,D

Where,

A = Entity Label of the Queue Block.

B = Upper Limit of the first frequency class.

C = Size of frequency class.

D = Number of frequency classes.

ADVANCE Blocks

The only Block that can advance time in a GPSS simulation is the ADVANCE Block. When a Transaction arrives at an ADVANCE Block it is delayed by a specified amount of simulated time and then proceeds to the next Block. An ADVANCE Block calculates a time increment and places the entering Transaction on the Future Events Chain (FEC) for that amount of elapsed simulated time. Its syntax is:

ADVANCE A, B

Where,

A = the required mean time increment.

B = the optional time half-range or, if a function, the function modifier.

The time increment can be calculated in several ways. If only the A Operand is specified, it is evaluated and used as the time increment. If the A and B operands are present, and B does not specify a function, both A and B are evaluated numerically and a random number between A–B and A+B, inclusively, is used as the time increment. If B is an FN class SNA, called a Function Modifier, the evaluating B is multiplied by the result of evaluating the A Operand; the product is used as the time increment. If zero is calculated as the time increment (ADVANCE 0), the entering Transaction is placed on the Current Events Chain in front of priority peers. Such a Block behaves as a null operation.

Function Entities

GPSS Function Entities are used to return a value derived from some argument, such as a random number. Actually, any SNA may be used as an argument. A Function is defined by a FUNCTION Command followed by one or more Function Follower Statements.

NAME FUNCTION A,B; comment

NOTES

The A operand of the FUNCTION Statement specifies the argument, and the B operand is the catenation of the Function type and the number of data pairs to appear on the Function Follower Statements. It is the numbers, names, and/or SNAs in the Function Follower Statements that complete the definition of the function entity. There are 5 different types of function entities:

- Type C—Continuous valued Function. Performs a linear interpolation. A random argument is a special case.
- Type D—Discrete valued Function. Each argument value or probability mass is assigned an separate value. A random argument is a special case.
- Type E—Discrete, attribute valued Function. Each argument value or probability mass is assigned an SNA to be evaluated. A random argument is a special case.
- Type L—List valued Function. The argument value is used to determine the list position of the value to be returned.
- Type M—List valued Function. The argument value is used to determine the list position of the SNA. This SNA is evaluated and returned as the result of the function.

A Function used in operand 'B' of an ADVANCE or GENERATE Block is called a "Function Modifier". The double precision floating point result of the function is multiplied by the evaluated 'A' operand. The result is then used as the time increment required by the Block. The SNA associated with functions is:

FNEntnum—Function. Result of evaluating Function Entnum.

TRANSFER Blocks

A TRANSFER Block causes the active Transaction to jump to a new Block or Command location that is not necessarily the next sequential Block. It is one way to allow non-sequential operation in a simulation. A Transfer Block has 4 operands (A,B,C,D):

A = One of 9 Transfer Block modes.

B = Block number or location.

C = Block number or location.

D = Block number increment for ALL Mode.

Unconditional Mode: A is blank. The active transaction always jumps to the location specified by the B Operand. A Transaction is never refused entry by a TRANSFER Block. If a Transaction becomes blocked by refusal of the destination Block it remains in the TRANSFER Block.

Fractional Mode: A is a probability (number between 0 and 1). The active transaction jumps to the location specified by the C operand

with probability A and to the location specified by the B operand with probability 1 - A. If B is blank the transaction goes to the next sequential Block (NSB). This is the most common use of a Transfer Block.

BOTH Mode: When the A Operand is BOTH the Block specified by the B Operand is tested. If it admits the transaction, then the location specified by the B Operand is the next transaction destination. Else the Block at the C Operand is tested. If this Block admits the transaction, then the location specified by the C Operand is the next transaction destination. If neither block admits the transaction, the Transaction waits at the TRANSFER Block until one of the Block specified by the B or C Operand admits it.

ALL Mode: When the A Operand is ALL, all the Blocks between that specified by the B Operand and the C Operand are tested in sequence. The first Block to admit the transaction will be the transactions destination. If no Block admits the Transaction, it stays in the TRANSFER Block until it can enter one.

PICK Mode: When the A Operand is PICK, the destination of the transaction is chosen randomly from the sequence of Blocks between those specified by the B Operand and the C Operand inclusively.

Function Mode: When the A Operand is FN the destination is chosen by evaluating a function entity, specified in B, and adding an optional increment specified in C. If C is not specified, the location specified by the B Operand is the location of the new destination.

Parameter Mode: When the A Operand is P the active Transaction jumps to a location calculated from the sum of a Parameter value named or numbered in the B Operand and the value of Operand C. If C is not specified, the Parameter value is the location of the new destination.

Subroutine Mode: When the A Operand is SBR he Active Transaction always jumps to the location specified by the B Operand. The location of the transfer Block is placed in the Parameter specified in Operand C. To return from the subroutine, use a TRANSFER Block in Parameter Mode as shown above.

Simultaneous Mode: When the A Operand is SIM the Active Transaction jumps to one of two locations depending on the Delay Indicator of the Transaction. If the Delay Indicator is set, the Transaction jumps to the location specified by the C Operand and the Delay Indicator is reset (turned off). If the Delay Indicator is reset (off), the Transaction jumps to the location specified by the B Operand. The Delay Indicator of a Transaction is set when the Transaction is refused by a Block. The Delay Indicator remains set until the Transaction enters a Simultaneous Mode TRANSFER Block.

NOTES

Test Blocks

A TEST Block compares values, normally SNAs, and controls the destination of the Active Transaction based on the result of the comparison. It is another way to allow non-sequential operation of the simulation. The syntax of a Test Block is:

TEST O A,B,C

Where,

O = Relational operator. It shows the relationship of Operand A to Operand B for a successful test. The operator must be E, G, GE, L, LE, or NE. Also sometimes referred as an auxiliary verb that is usually separated from the verb TEST by a single space whereas the Operands are usually separated from the verb by a tab.

A = Test value.

B = Reference value.

C = Destination Block number.

The relational operator is required. It may be E, G, GE, L, LE, or NE. The usual use of the Test Block is as follows:

If the test is successful the next Block to which the transaction is transferred is given by the C Operand; if not the Transaction proceeds to the next sequential Block. The successful tests are defined as follows:

E = the value of Operand A must be equal to the value of Operand B.

G = the value of Operand A must be greater than the value of Operand B.

GE = the value of Operand A must be greater than or equal to the value of Operand B.

L = the value of Operand A must be less than the value of Operand B.

LE = the value of Operand A must be less than or equal to the value of Operand B.

NE = the value of Operand A must be unequal to the value of Operand B.

Commands

Commands are GPSS statements that are immediately executed when a Transaction gets to the command. Some Commands are used to direct or modify the compilation and/or the execution of the simulation without having a Transaction arrive at the Command. They have the same syntax as Blocks.

NOTES

GPSS World

GPSS World is based on the seminal language of computer simulation, GPSS, which stands for General Purpose Simulation System. This language was developed primarily by Geoffrey Gordon at IBM around 1960, and has contributed important concepts to every commercial discrete event Computer Simulation Language developed ever since. GPSS World is a direct descendent of GPSS/PC, an early implementation of GPSS for personal computers. Since it was introduction in 1984, GPSS/PC and its successors have saved thousands of users millions of dollars. Now, the Windows implementation of GPSS World extends these capabilities into an Internet aware environment.

GPSS World is designed to deliver answers quickly and reliably, with the least effort, achieving the highest reliability of results. Consistent with these objectives, visualization of running simulations is highly stylized and a default statistical treatment is built in. This approach means that animations are "free" requiring no additional effort to produce, but are not photo-realistic. GPSS World's forte is transparency, not photo-realism. Third party animation systems are available which can provide pictorial animations based on GPSS World simulations.

GPSS World was designed to address these issues. Its visual nature allows the internal mechanisms of models to be revealed and captured. Its interactivity allows one to explore and manipulate simulations. Its built-in data analysis facility can calculate confidence intervals and an Analysis of Variance easily. And now, it can even create and run sophisticated screening and optimization experiments automatically, with relatively little effort from you. Most systems can be modeled in any of several ways using GPSS World. Usually only a small subset of the features available is needed to be used. However, the greatest proficiency requires familiarity with all that GPSS World has to offer. This manual is the primary source of that information.

GPSS World is object oriented. Its inhabitants include Model Objects which are used to create Simulation Objects. Simulation Objects, in turn, are used to play out simulations and create Report Objects. Finally, Text Objects can be used as Include-files to support code sharing and a user source code library and they are often used as files which can be read from or written to by the simulation.

In Version 4 of GPSS World, the PLUS Language has been extended for definition of Experiments. This powerful feature permits programmable control and can even be based on simulation results. Thus, completely automatic operation is possible including the exploration of response surfaces. PLUS Experiments, which are invoked by the CONDUCT Command, can be used to control the running of simulations over a parameter space. Experiments can be created with minimal involvement

NOTES

on your part. GPSS World now includes two new powerful experiment generators that can be accessed through the Edit Menu of the Main Window. The Screening Experiment Generator will create a fractional factorial experiment under your direction, and insert its PLUS source code into your model. Similarly, the Optimizing experiment generator will insert a sophisticated response surface exploration experiment that searches for minimum or maximum yields. The CONDUCT Command necessary to initiate the experiments is by default loaded into a Function Key. To execute a Screening or Optimizing Experiment and analyze the results, all you have to do is press a button.

GPSS World has comprehensive discrete and continuous modeling capabilities. The tightly knit continuous modeling feature allows for easy transition between the continuous and the discrete phases. From the continuous phase thresholds can be established that trigger the creation of Transactions for the discrete phase. Conversely, the INTEGRATION Block and INTEGRATE Command, control the processing in the continuous phase.

Several new GPSS Blocks have been added to cover the control of integrations, Transaction rescheduling, changes to Assembly Sets, user-defined PLUS Blocks, and Data Streams. Data types now include integer and real numeric values, as well as strings. Each type is automatically coerced to the required form, as needed. The new data type "UNSPECIFIED" now is available to indicate the absence of valid data, such as that from a missing run of an experiment. Even matrix structures have been improved. They can now incorporate up to 6 dimensions. The new multiway ANOVA Library Procedure analyzes data from a "Result Matrix", which is nothing more than a GPSS Matrix Entity where the yields of an experiment have been stored in a conventional way.GPSS World is easy to operate. A full screen text editor can be used in any of the Text Windows. Even the Journal Window and Reports can be customized and annotated.

GPSS World provides for a set of snapshots, as well. These are advanced features intended for professionals who need detailed knowledge of microstates of the simulation. Static snapshots can be taken of any Transaction, the Future Events Chain, the Current Events Chain, or the membership of the Numeric and Transaction Groups. GPSS World is highly interactive. All Model Statements can be used interactively.

Simulations can communicate with the outside world. You can now use five new GPSS Blocks and/or PLUS Procedures to manipulate Data Streams. They are OPEN, CLOSE, READ, WRITE, and SEEK. The last of these, SEEK, provides for direct access into a database. Data Streams have many purposes. You can use them to access data files, to create Result Files and custom reports, to access internal data directly. There are now PLUS Library Routines with the same

names that perform nearly the same functions. That means that complex I/O operations can now be handled inside User-created PLUS Procedures. GPSS World simulations can even communicate with other products by direct invocation. The PLUS Procedure Library now contains the Call(), Call_Integer(), Call_String(), and the Call_Real() procedures which can invoke external functions existing on your system in executable files (i.e. EXE and DLL files).

The analysis of results is easy in GPSS World. It has facilities that support the capture and printing of graphical windows. The Journal Window records the activities associated with the Simulation Object. An automatic Report numbering system ensures the safe keeping of each Standard Report. The new ANOVA Library Procedure can perform a complete multiway Analysis of Variance when passed a Result Matrix. That is all there is to the generation of an ANOVA table and a set of Confidence Intervals. And now, the Automatic Experiment Generators can create sophisticated screening and optimizing experiments based on dialog-window input. The basic analyses of these experiments are done automatically and reported in the Journal Window.

5.11 OBJECT-ORIENTED SIMULATION

Object Oriented Simulation (OOS) can be considered as a special case of Object Oriented Programming (OOP). Some principles of OOP like existence of a varying number of instances of interfering objects have been in standard use in simulation environment for a long time, often using other terminology. The Simula language is the first true object oriented language. Being more than 30 years old, it still has most (and all important) mechanisms and principles of OOP. Some things like classes, inheritance, virtual methods, etc., have been defined in Simula long time before; they were rediscovered by the Object Oriented Programming boom in recent years.

Object-oriented simulation offers excellent tools for treating models of complex dynamic systems. First of all, we must decide when the system is really complex. Obviously, a system described by a huge set of equations is not necessarily complex. We can solve on a computer set of thousands of simultaneous differential equations, but this does not mean that the model we solve is complex. On the other hand, a system may result to be complex even if the equations are apparently simple, but the system components have very distinct dynamic behavior or if they are of different kind. We say that a dynamic system is complex, if it has multiple components that reveal different dynamic properties. This may occur, for example, when all system components are continuous with concentrated parameters, but the model includes very fast and very slow parts.

NOTES

Other example is a system where discrete parts interact with continuous sub models of different speed and different kind, for example an electronic circuit that contains integrated circuits as well as electro-mechanical parts such as relays and motors. In other words, the model complexity has little to do with the model size. Using object-oriented approach we can simulate complex systems creating objects that simulate system sub models and run concurrently. The idea is to launch a set of objects and to cocrdinate them by other object that only connects the sub models and controls a general (global) interaction rules. Thus, a sub model of very different kind can run and interact in the same simulation program; some of them with integration step thousands of times smaller than others and some of them being discrete or combined.

5.12 SIMULATION PACKAGES

The vast amount of simulation software are available for the new users. There are several things that make an ideal simulation package. Some are properties of the package, such as support, reactivity to bug notification, interface, etc. Some are properties of the user, such as their needs, their level of expertise, etc. For these reasons asking which package is best is a sudden failure of judgment. The first question to ask is for what purpose you need the software? Is it for education, teaching, student-projects or research?

Animation in systems simulation is a useful tool and a user friendly tool must posses it. Most graphically based software packages have default animation. This is quite useful for model debugging, validation, and verification. This type of animation comes with little or no additional effort and gives the modeler additional insight into how the model works. However, it augments the modeling tools available. The more realistic animation presents qualities that intend to be useful to the decision-maker in implementing the developed simulation model. There are also, good model management tools. Some tools have been developed which combined a database with simulation to store models, data, animations, and results. However, there is not one product that provides all of those capabilities.

Examples of some simulation packages are CSIM18, JSSim(JavaScript Simulation), Warped, BaseSim, OpEMCSS, TomasWeb, DEx, SimulAr, ACSL Sim, DESIRE, FreeMat, Scicos, etc.

5.13 IMPORTANT SIMULATION LANGUAGES

NOTES

ACSL Sim : ACSL (a language for simulation of continuous system) developed by AEgis Research into ACSL Sim, a tool for the simulation of continuous systems.

DSDS+ : The Data Systems Dynamic Simulator Plus (DSDS+) is a discrete-event-based simulator that eases the difficulties associated with simulation of high-data-rate, end-to-end systems.

JiST : JiST is a high-performance discrete event simulation engine that runs over a standard Java virtual machine. It is a prototype of a new general-purpose approach to building discrete event simulators, called virtual machine-based simulation, which unifies the traditional systems and language-based simulator designs. JiST is developed by Cornell Research Foundation, Inc. and it is free for non commercial use.

MathCore : MathCore AB offers two main products: MathCore C++, an add-on to the well known Mathematical environment that compiles a subset of Mathematical into highly efficient C++ code. MathCode C++ provides a platform for rapid development of simulations and other expensive computations. MathModelica is an implementation of Modelica in Mathematical. MathModelica permits object oriented design of physical systems for simulation and visual programming using a graphic editor. MathModelica integrates documentation, runnable code, graphic connection diagrams and mathematical formulae in Mathematical notebooks.

MODSIM III : A language for simulation both objects and process oriented. MODSIM was recently sold to Compuware. CACI has migrated MODSIM users to SIMSCRIPT.

Parsec : Parsec a C-based simulation language, is used for sequential and parallel execution of discrete-event simulation models. It can also be used as a parallel programming language.

Pasion : Pasion is an object-oriented simulation language. The language has a process/event structure. Pasion source code is translated in Pascal, compatible with Delphi v3 or later. It can be used to model Queueing models, Continuous processes and allows the use of the Bond graphs paradigm.

SIMPLE_1 : SIMPLE_1 supports modeling discrete and continuous systems world views using a network modeling orientation. Features of the language include the ability of the user to declare variables and statistics requirements, perform input/output operations on files and to animate simulation results in real time easily utilizing built in language features. SIMPLE_1 utilizes a repetitive approach to run control to facilitate goal seeking modeling and run length definition based on model behavior.

NOTES

SimPy : SimPy (Simulation in Python) is an object-oriented, process-based discrete-event simulation language based on standard Python and released under the GNU GPL. It provides the modeler with components of a simulation model including processes, for active components like customers, messages, and vehicles, and resources, for passive components that form limited capacity congestion points like servers, checkout counters, and tunnels. It also provides monitor variables to aid in gathering statistics. Random variates are provided by the standard Python random module. SimPy comes with data collection capabilities, GUI and plotting packages. It can be easily interfaced to other packages, such as plotting, statistics, GUI, spreadsheets, and data bases. SimPy is under active development by an international development team.

WinSAAM : The WinSAAM modeling system is a Windows-based version of the SAAM and Consam modeling systems. It has been developed under the auspices of the Laboratory of Experimental and Computational Biology, of the Division of the Cancer Biology and Diagnosis, of the National Cancer Institute. It is provided freely for use in scientific research.

XMLlab : XMLlab is an XML-based simulation authoring environment. The proposed description language allows to describe mathematical objects such as systems of ordinary differential equations, systems of non-linear equations, partial differential equations in two dimensions, or simple curves and surfaces. It also allows describing the parameters on which these objects depend. The simulation is written in XML, according to the DTD file, and then transformed into a Scilab-executable file.

5.14 SIMULATION LANGUAGES VS. PROGRAMMING LANGUAGES

Following are the major differences between simulation languages and general programming languages:

1. Simulation languages can be used for only simulating a system or a model, while general programming languages can be used for any purpose.
2. Although we can use general purpose language for system simulation but the programmer may face problem of lengthy coding, but in case of simulation languages a wide variety of tools are available to avoid lengthy coding.
3. Most of the simulation environment have graphical user interface but only few general purpose languages have graphical user interface. For example, C/C++, Java, Pascal, FORTRAN has no graphical user interface.

NOTES

SUMMARY

- Within the past several years, significant developments in simulation software have taken place: new simulation languages have been developed; new software packages have been developed for use in conjunction with simulation for purposes other than building models; new features have been added to existing languages; vendors new to the simulation community have marketed implementations of existing software packages; simulation *environments, comprising integrated collections of simulation software* tools have been built. As a consequence of these developments, those readers whose perceptions of simulation software are several years old should consider themselves out of date. Enormous amounts of time and energy are presently being expended on research and development of simulation software. Thus, we can expect dramatic changes to take place in the near future. Simulation software of the 1990's will be as far removed from present software as present software is removed from building models "from scratch" in languages such as FORTRAN.
- In this unit we have described new techniques for simulating systems with complex structures by use of a digital computer, as well as the requirements of tools (simulation languages) to cope with the problem in a user-friendly way. Emphasis is given to a general applicability of the software, that is, the described software is meant to be able to handle broad classes of problems in a sub-optimal way rather than to *be able to treat any specific application problem in a truly optimal* manner. The increased software robustness and the highly reduced costs in coding any application problem compensate, however, for the sacrifice of efficiency in executing a particular simulation project. Although many problems of numerical mathematics and information processing which are discussed in this chapter had to be considered and solved, the approach to them has been from the viewpoint of an engineer rather than from that of a mathematician.

GLOSSARY

- **PERT:** Is a network model that allows for randomness in activity completion times.
- **CPM:** Is a network model for project management which uses a fixed time estimate for each activity.
- **WBS:** A detailed project shedule, can be defined with the use of project network diagrams for the entire project.
- **PERT:** Is a paint that marks the start or event completion of one (or more) task.
- **Optimistic time:** The shortest time in which the activity can be completed.
- **MODSIM III**: An object and a process oriented language for simulation.

NOTES

REVIEW QUESTIONS

1. Give a comparative study of PERT, CPM and Gantt Charts.
2. Explain the advantages and disadvantages of Gantt chart.
3. What do you understand by Forward pass and backward pass in activity networks?
4. What is the contribution of PERT, CPM and Gantt charts in system simulation?
5. A project has characteristics given by following table:

Activity	*Preceding activity*	*Duration (Week)*
A	None	5
B	A	2
C	A	6
D	B	12
E	D	10
F	D	9
G	D	5
H	B	9
I	C, E	1
J	G	2
K	F, I, J	3
L	K	9
M	H,G	7
N	M	8

Determine the following:

(*i*) PERT network for this project.

(*ii*) Find various paths and the critical path as well as the project completion time.

(*iii*) Prepare the activity schedule showing the ES, EF, LS, and LF.

6. A company planning a project to introduce a new product, has listed the following necessary activities:

Activity	*Preceding activity*	*Expected time (Week)*
A	None	6
B	None	3
C	A	5
D	A	4

E	A	3
F	C	3
G	D	5
H	B, D, E	5
I	H	2
J	I, G, F	3

Determine the following:

(*i*) Draw the critical path network for the project and determine the critical path and its duration.

(*ii*) If the start of the activity B is delayed 2 weeks, activity E by 1 week and activity G by 2 weeks, how the total project time will be affected?

7. What do you understand by simulation languages? Give the differences between continuous and discrete system simulation languages.
8. What is advanced continuous simulation language? What are its major features?
9. Give a comparative study of SIMULA and SIMSCRIPT.
10. Explain the object oriented features of SIMULA language.
11. Give the differences between simulation packages and simulation languages.
12. Write short notes on following:
    (*i*) Online simulation
    (*ii*) Visual simulation tools
    (*iii*) Distributed simulation tools.
13. Explain the factors used in the selection of a simulation language for a particular system simulation.
14. Explain the differences between simulation languages and general purpose programming languages.

FURTHER READINGS

...ling and Simulation Concepts', by Rajinder Kumar, Anil Kumar, ...Kumar, Chandra Shekher Yadav. University Science Press.