Modelling a system

A model is the body of information about a system gathered for the purpose of studying the system.

The purpose of the study will determine the nature of the information that should be collected. There is no unique model for a system.

A model contains only the essentials of the real system. Those aspects of the system which do not contribute significantly to system behaviour are excluded.
Studying a system

Experiment with actual system

Experiment with a model

Physical model

Mathematical (logical) model

Analytical solution

Simulation

Analytical modelling vs simulation

- Mathematical model: represents a system in terms of logical and quantitative relationships that are then manipulated through mathematical equations to see how the model reacts to different parameters

- Solutions to a model
  - Analytical Solution: If exact, analytical (or closed – form) solution exist (e.g., some Queuing models)
  - Simulation: If analytical solutions are extraordinarily complex, and requires vast computing resources.

- Usually we do not simulate when the system can be solved analytically.
Steps in designing a system

- Develop a simulation model of the system
- Experiment with the model (by simulation)  
  - $\Rightarrow$ refinement of the model
- Develop a prototype and experiment with it  
  - $\Rightarrow$ refinement of the model
- Implement and test the system  
  - $\Rightarrow$ refinement of the model

Classification of Models

- Static vs dynamic models
  - A system that does not vary with time is static whereas one that varies is dynamic.
  - static: time invariant, e.g., calculation of an area by simulation
  - dynamic: time dependent, e.g., $M/M/1$
### deterministic vs stochastic models

**Deterministic:** no random variables
- Deterministic models have a known set of inputs which will result in a unique set of outputs.
  - e.g. patients arriving at a clinic at scheduled appointment time (deterministic arrivals)

**Stochastic** (non-deterministic or probabilistic)
- Models have one or more random variables
  - e.g. Bank - random customer inter-arrival and service times.

---

### Monte Carlo Simulation

- describe systems which are both stochastic and static
- can be employed to study complex phenomena that are insufficiently well understood to be amenable to more detailed cause/effect analysis, or when solutions to non-probabilistic models are too complicated or computationally too expensive.
- e.g. economic forecasting, risk analysis, reliability of nuclear power station, etc.
continuous vs discrete time models

- **Discrete time model**: states variables change at discrete points in time, e.g., queue length of a server
- **Continuous time model**: state variables may change continuously, e.g., temperature in a class room
  - the systems modelled are dynamic but may be either deterministic or stochastic
  - continuous simulation models are used extensively in mechanical, production and electrical engineering.
  - example: *the increase and decrease in water temperature is a continuous process which can be represented in a continuous simulation as a number of differential equations.*

Combined Discrete/Continuous models (Hybrid)

- some variables in the system model are discrete and some continuous
  - example: *unloading dock where tankers queue up to unload their oil through a pipeline*
  - discrete - tanker arrivals
  - continuous - flow of oil

◆ **NOTE**: Choice of simulation model is a function of the characteristics of the system and the objectives of the study.
What is a simulator?

A simulator is an implementation of a model (e.g., a computer program which mimics the behaviour of the real-world system).

Simulator output is a set of measurements concerning the observable reactions and performance of the system.

WARNING: Measurements are only estimates of what the real world system actually would be.

Output of a simulator

Why are the measurements some estimates?

Because only an abstraction of the real-world system is simulated!

Example

System: bank
Input
- Arrival of customers
- Service rate of the server
Output
- Average queuing time
- Average service time
Why simulation?

Typically, we simulate rather than experiment with the real world system because

- the system as yet does not exist.
- experimentation with the system is
  - too expensive
  - too time consuming
  - too dangerous.
- experimentation with the system is inappropriate, e.g. disaster planning

Advantages of Simulation

- Permits controlled experimentation
- Permits time compression so that policy decisions can be made, e.g. weather forecasting
- Permits the evaluation of operating performance prior to the implementation
- Permits the comparison of various operational alternatives without perturbing the real system
Disadvantages of Simulation

- May require large expenditure of time (manpower) in constructing and validating the model
- Numerous simulation runs are usually required – computationally expensive
- Hidden critical assumptions may cause the model to diverge from reality
- Model parameters may be difficult to initialise

Simulation life cycle

1. Real world system
2. Problem formulation
3. Conceptual model
4. Formal analysis
5. Output data analysis
6. Simulation analysis
7. Coding
8. Testing
9. Simulation experimentation
Simulation Life Cycle

PHASE 1
Definition of Problem and Objectives

1. Problem formulation
   - Setting of objectives and overall project plan

PHASE 2
Model Building and Data Collection

2. Model building
3. Data collection

4. Coding
   - Verified?
     - yes
     - no

5. Verified?
   - yes
   - no

6. Validated?
   - yes
   - no

PHASE 3
Simulation Experiments and Analysis

7. Experimental design
8. Production runs and analysis
   - More runs?
     - yes
     - no

PHASE 4
Documentation and Implementation of the Results

9. Document program and report results
10. Implementation

©2003, Rassul Ayani
Summary

- Simulation has a wide range of applications
- This course will be focused on modelling and simulation of dynamic, stochastic, and discrete systems

INTRODUCTION TO DISCRETE-EVENT SIMULATION (DES)

- Objectives
  - to understand different approaches to simulation modeling
- Outline
  - What is DES?
  - How do we advance simulation time?
    - fixed-increment time advance
    - next-event time advance
What means DES?

A discrete-event simulation models a system whose state may change only at discrete point in time.

- The assumption is that nothing relevant happens between successive state transitions.

- The system is composed of objects called entities that have certain properties called attributes.

- The system state is a collection of attributes or state variables that represent the entities of the system.

- An event is an instantaneous occurrence in time that may alter the state of the system.

Terminology

- An event initiates an activity, which is the length of time during which entities engage in some operations.

- Entities, attributes, events, activities and the interrelationships between these components are defined in the model of the system.

- Any objects and activities that lie outside of the system boundary, but can influence the system, make up the system environment.
Example: a single server system

- **entities**: customers; server
- **attributes of a customer**: service required
- **attributes of server**: server's skill (its service rate)
- **events**: arrival of a customer; departure of a customer
- **activities**: serving a customer, waiting for a new customer

![Diagram of a single server system]

---

Example: a manufacturing plant

- **entities**: machines, workers, bundles of material, machine repairmen
- **attributes of a machine**: its operation (hemming, sewing, pressing, etc.), its efficiency, its currently assigned worker.
- **attributes of a worker**: his skills, efficiency rating, etc.
- **activities**: sewing, hemming, inspection, packaging, etc.
- **events**: arrival of a bundle to a work-station for sewing, departure of a worker for lunch break

![Diagram of a manufacturing plant]
Advancement of simulation time

Fundamental to every simulation study is a mechanism to model the passage of time. Thus every model contains a variable called the internal clock, or the simulation clock.

How do we advance simulated time?
- fixed-increment time advance approach (synchronous model of time)
- next-event time advance approach (asynchronous model of time)

Fixed-increment (or time-stepped) approach
- clock is updated by the same time increment, $\Delta t$

After each clock update, all events that were scheduled to occur during this interval are identified. Events are considered to occur at the end of the interval and the system state are updated accordingly.

Useful to model system in which events occur only at intervals of some fixed length.

Disadvantages
- small $\Delta t$: wasteful scanning - no events happen during $\Delta t$
- big $\Delta t$: lost accuracy (it assumes that simulation state doesn't change between events)
Next-event time advance

- more commonly used approach
- time is advanced from the time of the current event to the time of the next scheduled event
- simulation skips over periods of inactivity. Saving in computer time to run simulation at the expense of increase programming effort.
- this method is called event-driven DES and is asynchronous as opposed to time-stepped approach which is synchronous

Future event list (FEL)

- Both the asynchronous and synchronous models assume that the events are chronologically ordered in time
- A calendar of events (also known as future events list, event list, event diary, calendar of events) contains all scheduled events, arranged in chronological time order. In the simulator, this is just a data structure, e.g. list, tree

\[ E_{t0} < E_{t1} < E_{t2} < E_{t3} < \ldots E_{tn} \]

- This generates a new event \( E_{t'n} \),
- This is placed at the appropriate Position in the event list using the Current time and Next event Time.

©2003, Rassul Ayani
Next-event time advance: Single-server Queue

- **Data structures:** future event-list (FEL), state vector (SV), clock (CL)
- **Repeat**
  - remove smallest event (with time t) from FEL
  - set CL=t
  - process the event (and modify the SV)
- **End_of_simulation**

An Algorithm for Event-Scheduling Approach (Single-server Queue)

**main (executive routine):**
1. set CLOCK = 0
2. set cumulative statistics to 0
3. define initial system state (queue empty, server idle)
4. generate the occurrence time of the first arrival and place in FEL
5. select the next event on FEL (arrival or departure event)
6. advance simulation CLOCK to time of next event
7. process this event (execute the corresponding event **routine**)
8. if not end-of-simulation goto step 5
COMPONENTS AND ORGANISATION OF A SIMULATION

**SYSTEM STATE:**
- the collection of state variables necessary to describe the system at a particular time.

**SIMULATION CLOCK:**
- a variable giving the current value of simulated time.

**EVENT LIST:**
- a list containing the next time when each type of event will occur

**Statistical Counters:**
- Variables used for storing statistical information about system performance

**Initialisation Routine:**
- A subprogram to initialise the simulation model at time zero.

**Timing Routine:**
- A subprogram that determines the next event from the event list and then advances the simulation clock to the time when that event is to occur.

COMPONENTS AND ORGANISATION of a DES

**Event Routine:**
- a subprogram that updates the system state when a particular type of event occurs (one event routine for each event type)

**Library Routines:**
- a set of subprograms used to generate random observations from probability distributions that were determined as part of the simulation model

**Report Generator:**
- a subprogram that computes estimates (from the statistical counters) of the desired measures of performance and produces a report when the simulation ends

**Main Program:**
- a subprogram that invokes the timing routine to determine the next event and then transfers control to the corresponding event routine to update the system state appropriately. The main program may also check for termination and invoke the report generator.
## Simulation of Queuing systems

### Why are queuing systems important?

<table>
<thead>
<tr>
<th>System</th>
<th>Customers</th>
<th>Server(s)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Hospital</td>
<td>Patients</td>
<td>Nurses</td>
</tr>
<tr>
<td>Computer</td>
<td>Jobs</td>
<td>CPU, disk drive</td>
</tr>
<tr>
<td>Telephone</td>
<td>Calls</td>
<td>Exchange</td>
</tr>
<tr>
<td>MRT</td>
<td>Passengers</td>
<td>trains</td>
</tr>
<tr>
<td>Airport</td>
<td>Airplanes</td>
<td>Runway</td>
</tr>
<tr>
<td>Reception desk</td>
<td>People</td>
<td>receptionist</td>
</tr>
<tr>
<td>Repair facility</td>
<td>Machines</td>
<td>repairperson</td>
</tr>
</tbody>
</table>

### Example: simulation of a single-server queuing system

- **measurement metrics:**
  - average delay in the queue
  - time-weighted-average number of customers in queue, \( q(n) \)
  - utilisation of server, \( u(n) \)

![A Single-Queue Single-Server System](image-url)
Simulation of a single-server system (cont’d)

- **Assumption**
  - interarrival times $A_1, A_2, \ldots$ are independent and identically distributed (IID) random variables. ("identically distributed" means that $A_i$ and $A_j$ have the same probability distribution for all $i$ and $j$)
  - service times $S_1, S_2, \ldots$ of the successive customers are IID random variables that are independent of the interarrival times

- **Service policy**
  - *First-in-first-out (FIFO)*

- **Event types:**
  - $A$: Arrival of a customer to the system
  - $D$: Departure of a customer

- **Event routines:**
  - one per event type

- **Termination criteria**
  - when $n$ customers receive service

---

Event-Scheduling Approach

- Data structures: future event-list (FEL), state vector(SV), clock(CL)
- Initialize the simulator
- Repeat
  - remove smallest event (with time $t_{min}$) from FEL
  - set $CL = t_{min}$
  - process the event (using corresponding event-routine)
- End_of_simulation
The Event-Scheduling Approach: Arrival routine

Arrival event routine:
1. if SERVER is idle then
   set server to busy
   update server idle time statistics
   generate the occurrence time of the departure event
   and place departure event in future events list
else (* server busy *)
   place the customer in QUEUE
   update QUEUE length statistics
2. generate the occurrence time of the next arrival and
   place arrival event in future events list
Departure Routine

Departure event

Eliminate departure

Is the queue empty?

No

Subtract 1 from the number in queue

Computer delay of customer entering service and gather statistics

Add 1 to the number of customer serviced

Schedule a departure event for this customer

Yes

Make the server idle

Return

Departure routine

1. if QUEUE is empty then
   set server to idle
   update SERVER idle time statistics
else
   remove the NEXT customer from the QUEUE
   update QUEUE waiting time statistics
   generate the occurrence time of the departure event
   and place departure event in future events list

2. destroy the current event
Simulation of a single-server system: an example

We assume:

- Interarrival times of customers are:
  \[ A_1 = 0.4, A_2 = 1.2, A_3 = 0.5, A_4 = 1.7, A_5 = 0.2, \ldots \]

- Service times of customers:
  \[ S_1 = 2.0, S_2 = 0.7, S_3 = 0.2, S_4 = 1.1, \ldots \]

Note that it is not necessary to declare what the time units are (minutes, hours, etc.) but only to be sure that all time quantities are expressed in the same units.

In a stochastic simulation, the \( A_i \)'s and the \( S_i \)'s would be generated from their corresponding probability distributions.

---

Arrival time: 0.4, 1.6, 2.1, 3.8, 4.0, 5.6, 5.8, 7.2 ...
Service time: 2, 0.7, 0.2, 1.1, ...

<table>
<thead>
<tr>
<th>Arrival events</th>
<th>A1</th>
<th>A2</th>
<th>A3</th>
<th>A4</th>
<th>A5</th>
<th>A6</th>
<th>A7</th>
<th>A8</th>
</tr>
</thead>
<tbody>
<tr>
<td>Time</td>
<td>0.4</td>
<td>1.6</td>
<td>2.1</td>
<td>3.8</td>
<td>4.0</td>
<td>5.6</td>
<td>5.8</td>
<td>7.2</td>
</tr>
</tbody>
</table>

\[ S_1 = 2.0 \]
\[ S_2 = 0.7 \]
\[ S_3 = 0.2 \]
\[ S_4 = 1.1 \]

<table>
<thead>
<tr>
<th>Time</th>
<th>Departure Events</th>
</tr>
</thead>
<tbody>
<tr>
<td>2.4</td>
<td>D1</td>
</tr>
<tr>
<td>3.1</td>
<td>D2</td>
</tr>
<tr>
<td>3.3</td>
<td>D3</td>
</tr>
<tr>
<td>4.9</td>
<td>D4</td>
</tr>
<tr>
<td>8.6</td>
<td>D5</td>
</tr>
</tbody>
</table>
Customers’ delay and the queue length

An estimate of the average delay in queue, \( d(n) \), is:

\[
\hat{d}(n) = \frac{\sum_{i=1}^{n} D_i}{n}
\]

where \( D_i \) denotes the time customer \( i \) has waited in the queue

An estimate of the time-weighted-average number of customers in queue, \( q(n) \), is:

\[
\hat{q}(n) = \frac{\int_{0}^{T(n)} Q(t) \, dt}{T(n)}
\]

\[
\hat{q}(n) = \frac{\sum_{i=1}^{n} i \cdot T_i}{T(n)}
\]

Where \( Q(t) \) denotes queue length at time \( t \), \( T_i \) is the total simulation time that the queue is of length \( i \), and \( T(n) \) is the time when \( n \) customers have been served.
Server utilisation

Utilisation of server, $u(n)$, is the proportion of time the server is busy:

- We define the “busy function” as:

\[
B(t) = \begin{cases} 
1 & \text{if the server is busy at time } t \\
0 & \text{if the server is idle at time } t 
\end{cases}
\]

\[
\hat{u}(n) = \frac{\int_0^{T(n)} B(t) \, dt}{T(n)}
\]
Simulation of M/M/1

M/M/1 is a single server queue with the following assumptions
- inter-arrival times are IID and have exponential distribution
- service time is IID and have exponential distribution

Measurement metrics
- expected average waiting time of a customer in the queue
- expected time-weighted average number of customers in queue
- expected average utilization of server

Stopping (termination) criteria
- simulation time = T or
- number of customers received service = M

SUMMARY of important points
- Simulation life cycle
- Time advancement in DES
- Event types and event routines in DES
- Measurement metrics in a simulation
void arrive(void) /* Arrival event function */
{
    float delay;
    /* schedule next arrival */
time_next_event = current_time + exp0n(mean_interarrival);
    /* check to see whether server is busy */
    if (server_status == BUSY)
    {
        /* server is busy, so increment no. of customers in queue */
        ++num_in_q;
        /* check to see whether an overflow condition exists */
        if (num_in_q > Q_LIMIT)
            /* the queue has overflowed, so stop the simulation */
            printf("\nOverflow of the array time_arrival at time %f\n", current_time);
            exit (2);
    }
    /* there is still room in the queue, so store the time of arrival of the
    arriving customer at the (new) end of time_arrival */
time_arrival[num_in_q] = current_time;
    /* schedule next arrival */
time_next_event = current_time + exp0n(mean_interarrival);
}
else