Simulation & Modeling

Course Code: BIT534CO
Year/ Semester: III/VI

Compiled by Shishir Ghimire

Credit Hours: 3hrs

Unit 6: Simulation Languages

Class Load : 10 hrs

TABLE OF CONTENTS

6. Simulation languages **[10 hours]**

6.1. Basic concept of Simulation tool
6.2. Discrete systems modeling and simulation
6.3. Continuous systems modeling and simulation
6.4. Structural, data and control statements, hybrid simulation
6.5. Feedback systems: typical applications

Simulation Tools:

- With different **abstraction levels,** different tools are used to perform and evaluate simulations.
- An important characteristic of a tool is how it supports **model building.** In many tools; one constructs networks of components whose local behavior is already known and already programmed into the tool.
- A very significant player in computer-systems design **at lower levels of abstraction is the VHDL language.**
- **VHDL (VHSIC-HDL): Very High Speed Integrated Circuit Hardware Description Language**
  - VHDL is a **hardware description language** used in electronic design automation to describe digital and **mixed-signal systems** such as field-programmable **gate** arrays and **integrated circuits.**
  - VHDL is a rich language, full both of **constructs** specific to digital systems and the constructs one expects to find in a procedural programming language.

Simulation Tools:

- **VDHL (Contd ...) :**
  - The language promotes **user-defined programmed behavior**.
  - VHDL is also innovative in its use of **abstract interfaces** which different "architectures" at different levels of abstraction may be attached.
  - VHDL is widely used in the **electrical engineering community** but is hardly used outside of it.
- **Disadvantage of VHDL**
  - VHDL is a big language, requiring a substantive **VHDL compiler**, and **vendors** typically target the commercial market at prices that **exclude** academic research.
- **Advantage of VHDL**
  - The relative rigidity of the programming model makes possible **graphical model building**, thereby raising the whole model-building endeavor up to a higher level of abstraction.
  - Some tools have so much preprogrammed functionality that it is to design and run a model **without writing a single line of computer code**.

Simulation Tools:

There are various simulation tools available in market. They are broadly divided into 2 types:

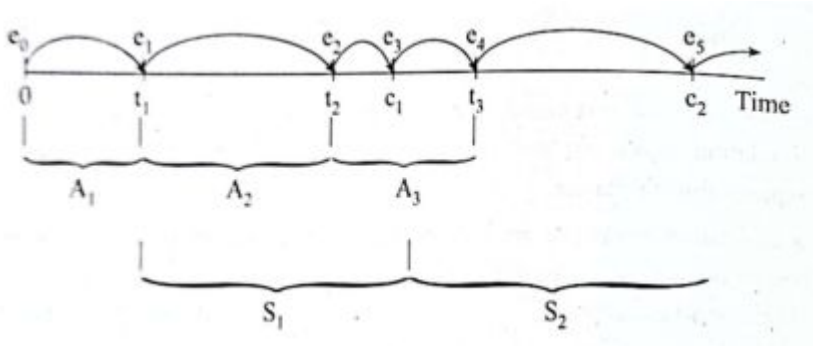
1. **Event Oriented Simulator**
2. **Process Oriented Simulator**

Simulation Tools > Event Oriented :

- The Event Oriented simulation model is one that is essentially an **event driven** finite state machine. The simulation **modeler concentrates on events and how they affect the state of the simulated system.**
- The simulation programmer **defines events** and then writes **routines which are invoked** as each kind of event occurs.
- The semantics of the program expressing a model are the **semantics of the programming language**; there is **no hidden activity** to stack management or thread scheduling.
- Simulated time may pass between the events.
- Usually, a **priority queue** will be used.
- Discrete event simulation concerns the modeling of a system by a representation in which the state of the **system changes at discrete time points** known as events.
- An **event is a time point** where the states of the **system changes or other major** things happen (such as the start and the end of the simulation).

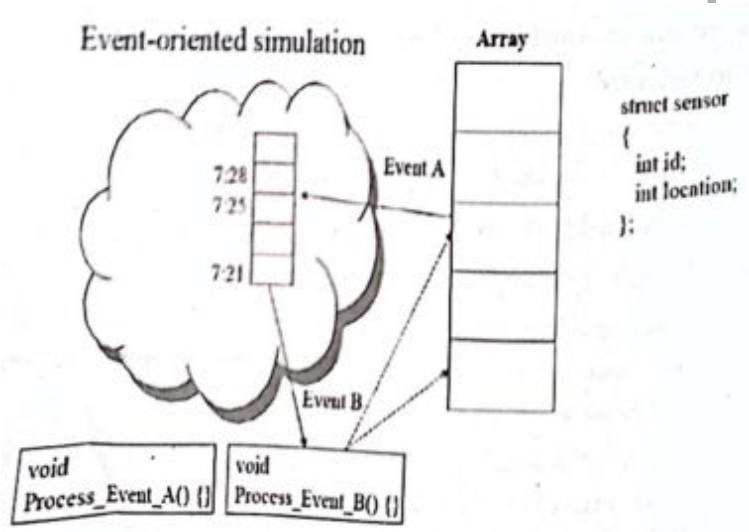
Simulation Tools > Event Oriented :

- An event-oriented simulator is much **simpler to design and implement** using a general programming language.
- The implementation requirements of an object-oriented event-oriented approach are **much less delicate** than those of a **threaded simulator**, and the amount of simulator overhead involved in delivering an event to an object is considerably less than the cost of a **context switch in a threaded system**.
- For these reasons, most of the simulators written from scratch take the **event-oriented view**.
- Example: When simulating GI/GI/1 queue, the two major events are customer arrival and customer departure.



Simulation Tools > Event Oriented :

- An arrival event **increases** the number of customer in **queue by 1** (if the server is busy) or changes the state of the server to busy (if the server was idle).
- A **departure event** decreases the number of customer in queue by 1 or changes the server state to idle.
- Simulated time is measured through the simulation clock.
- The simulation clock is initialized at 0 and the times of occurrence of future events are determined.
- The simulation clock is then advanced sequentially to the time of the "next" event.
- The system is observed at event times and statistics of performance measures are collected.



Simulation Tools > Process Oriented :

- A process-oriented view implies that the tool must support **separately schedulable threads of control**. Fundamentally, a "thread" is a **separately schedulable unit** of execution control, implemented as part of a single executing process.
- Process oriented simulation (applied on "high level" software such as Arena) allows the user to **skip** the tedious **event-driven details of discrete event simulation.**
- It is based on viewing the simulation in terms of the **experience of entities that flow through the system**.
- The logic behind process oriented simulation is **similar to constructing a flowchart** for the flow of an entity in the system.
- The thread yields by executing a statement whose semantics mean "**suspend this thread until the following condition occurs**"
- The **hold statement** is a classic example of this, it specifies how long in simulated time the thread suspends.

Simulation Tools > Process Oriented :

- For example, imagine you are a customer in the **bank example**, then your experience will be as follows:
  - Create yourself (i.e. arrive to the system).
  - Note your arrival time.
  - Wait in queue if the server is busy.
  - Once the server is idle, move out of the queue and “seize” the server.
  - Note your start of service time and estimate your waiting time in queue.
  - Stay in service (i.e. “delay” yourself) for an amount of time equal to your service time.
  - “Release” the server.
  - “Dispose” of your bank visit (leave the system).
- Process oriented simulation is a way to think about the system and simulate it in a **software like Arena**.
- The programmer defines the **processes** (entities, transaction etc) and model in terms of **interacting processes**.

Simulation Tools > Process Oriented :

Example: CSIM Stack in Threaded Simulation

- Figure 9.2 illustrates a **sequence of procedures** and the **stack of frames** present during the first call to hold. This call suspends the thread, and when the thread resumes execution, it needs access to all the **data stored** on the stack at the **time of suspension**.
- To support this, the threading mechanism must **save** and later **restore the relevant portion** of the stack.
- CSIM uses a single shared stack space for all threads during execution. Here's how it works:
  - The **first procedure** frame in any thread always starts at the same memory location.
  - When a thread suspends, the CSIM kernel copies the active portion of the stack into a **separate memory area** associated with that thread.
  - When the thread is **ready to resume**, the kernel:
    - **Copies the saved stack** back into the fixed memory location.
    - **Restores** the saved machine registers.
    - Hands control back to the **runtime system**, which continues execution as normal.

```
Void procA(int a)
{
create("thread");
int x;
procB(x=2*a);
procB(2*x);
}
void procB(int b)
{
int x;
procC(x=2*b);
procC(2*x);
}
void procC(int c)
{
hold(c);
}
```

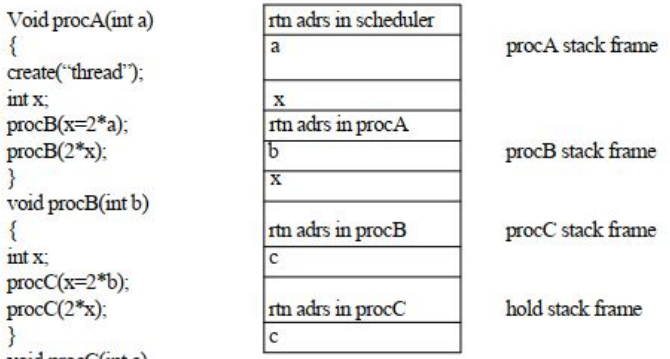


Fig. 9.2 Runtime stack in threaded simulation

Simulation Tools:

Characteristic	Event-Oriented Simulator	Process-Oriented Simulator
Modeler Focus	Changes in **System State** via instantaneous **Events**.	**Entities** and their **Life Cycle/Workflow** (Process).
Logic	**Fragmented** (scheduling is scattered across event routines).	**Coherent** (logic follows the path of a single entity).
Primary Data Structure	**Future Event List (FEL)**.	**Separately Schedulable Processes/Threads**.
Modeling Ease	**Lower** (Requires defining many discrete events).	**Higher** (Allows skipping tedious event details).
Execution Speed	**Faster/More Efficient** (Low overhead, only a clock advance and event list management).	**Slower/Less Efficient** (Higher overhead due to context switching).

Simulation Tools:

Example of Simulation Tools :

- **GPSS** – mostly applicable to queuing situations
- **SIMAN & SLAM II** – appropriate for simulating the manufacturing and material handling system
- **AnyLogic** - a multi-method simulation modelling tool for business and science
- **PTV Vissim** - a microscopic and mesoscopic traffic flow simulation software.
- **SimPy** - an open-source discrete-event simulation package based on Python.
- **OpenSim** - an open-source software system for biomechanical modelling.
- **NetLogo** - an open-source multi-agent simulation software.
- **Flood Modeller** - hydraulic simulation software, used to model potential flooding risk for engineering purposes.
- **MATLAB** - a programming, modelling and simulation tool developed by MathWorks.
- **VSim** - a multiphysics simulation software tool designed to run computationally intensive electromagnetic, electrostatic, and plasma simulations.

Simulation Language:

- A simulation language is a **special purpose language structured to meet the programming requirements of the simulation models** of a specific class of situations.

- The analyst develops the simulation **model**, employing this special purpose **language**, which is applicable over a class of applications.

Simulation Language > Types :

There are essentially **three types of languages** one can use for simulation:

1. **General Purpose Programming Language:**
   - ➢ **such as Java, C++, C**
2. **Simulation Programming Language:**
   - ➢ **such as GPSS, Simul**
3. **Simulation Tools or Environment or Software:**
   - ➢ **such as Arena, AutoMod**

Simulation Language > Types :

- **General Purpose Programming Language (GPPL):**
  - GPPL such as **Java, C++, C** can be used to developed the simulation model of real system as a computer program and can be used for the experimentation purpose.
  - It may be the **difficult and time consuming task** since the GPPL may not support directly tools and function required by a particular simulation model.
  - **In such a case,** we have to use specially designed language for simulation.
  - **Advantages:** They tend to be **very fast**, there are **few limitations** on what can be done, programmers are easily found, and simulations can be done on a number of different computers (due to standardization of language).
  - **Disadvantages:** **Expensive** and **need extensive knowledge** of simulation implementation which is generally very large and intricate programs. The time needed to design, code, and verify such a system may be overwhelming.

Simulation Language > Types :

- **Simulation Programming Languages:**
  - Simulation programming languages are **specifically designed for simulation**, such as **GPSS and SIMAN**.
  - They provide **routines** and **data structures**, such as **queue manipulation**, to facilitate simulation.
  - The major **disadvantage** of simulation languages is that they are **highly specialized** and not **readily extensible**.
  - **For example**, incorporating a network optimizer in a transportation simulator written in SIMAN may require linking to FORTRAN programs. This is both **inconvenient** and **inefficient**.

Simulation Language:

- **Simulation Tools or Environment or Software:**
  - There are **simulation environments or IDEs** such as **Arena, AutoMod**, etc.
  - These products include common characteristics such as a **graphical user interface** and **an environment** that supports all aspects of a simulation study.
  - **Some other examples are:**
    - ✓ ACSL, APROS, ARTIFEX, C++ SIM, CSIM, CallSim, FluidFlow, GPSS, Gepasi, JavSim, MJX, MedModel, Mesquite, Multiverse, NETWORK, OPNET Modeler, POSES++, Simulat8, Powersim, QUEST, REAL, SHIFT, SIMPLE++, SIMSCRIPT, SLAM, SMPL, SimBank, SimPlusPlus, TIERRA, Witness, SIMNON, VISSIM, and Javasim.

Simulation Language:

The **important features** of simulation language can be identified as follows:

- **Generation** of large streams of **random numbers**
- Generation of random variates from a large number of probability distributions
- Determining the **length of simulation run and length of wrapping (covering) up period**
- Advancing the **simulation clock**
- Scanning the event list to determine the **next earliest event** to occur
- **Collecting data**
- **Analysing data** and setting **confidence intervals**

Simulation Language > Objectives :

These common features, which have to be invariably (in every case) modelled in all simulations, are perhaps the cause of the development of special simulation software. Simulation software packages are designed to meet the **following objectives:**

- ***To conveniently describe the elements***, which commonly appear in simulation, such as the generation of random variates.
- ***Flexibility of changing the design configuration of the system*** so as to consider alternate configuration.
- ***Internal timing and control mechanism***, for book keeping of the vital information during the simulation run.
- ***To obtain conveniently*****,** the data and statistics on the behaviour of the system.

Simulation Language:

Merits of Simulation Language

- Since most of the features to be programmed are in-built, simulation languages take **comparatively less programming time and effort.**
- Since simulation language consists of blocks, specially constructed to simulate the common features, they **provide a natural framework for simulation modelling**.
- The simulation models coded in simulation language **can easily be changed and modified.**
- **The error detection and analysis is done automatically** in simulation languages.
- The simulation models developed in simulation languages, especially the specific application packages, called simulators, **are very easy to use.**

Simulation Language > Classes :

Can be categorized into two board classes:

1. Discrete System Simulation Language
2. Continuous System Simulation Language

Simulation Language > Classes :

- **Discrete Simulation Language**
  - A number of **programming languages have been developed** to simplify the task of writing discrete system simulation programs.
  - These programs **include** a **language with which to describe the system** and a **programming system** that **will establish a system image and execute a simulation algorithm**.
  - Each language is based **on a set of concepts used for describing systems**.
  - The word ***world-view*** has come to be used to describe this aspect of simulation programs. The user of the program must learn the **world-view of the particular language he/she is using** and be able to describe the system in those terms.
  - ***GPSS: A discrete system simulation language.***

Simulation Language > Classes :

- **Continuous System Simulation Language (CSSL):**
  - CSSL uses the familiar **statement type of input** for digital computers, allowing a program to be programmed **directly from the equations of a mathematical model**, rather than requiring the equations to be broken into **functional elements.**
  - CSSL **includes a variety of algebraic and logical expressions** to describe the relation between variables.
  - Digital analog simulator **have restriction** on representing the functions of an analog or hybrid computers. Continuous system simulation language **overcome three restriction.**
  - CSSLS include **macros or sub routine** that perform the function of specific analog elements. It allow the users to define special purpose elements that correspond to operations that are particularly in specific type of applications.
  - Several implementations of CSSL have been published. One particular CSSL that illustrates the nature of these languages is the **Continuous System Modeling Program** (CSMP, version 3), i.e., CSMP III.

Structural, Data & Control Statements:

- A simulation language provides **a specific set of commands, or "statements,"** that allow a modeler to define the **system's static architecture**, its properties, and its dynamic behavior. These statements are the **fundamental building blocks** of a simulation model.
- CSSLS include variety of algebraic and logical expression to describe the relations between variables. CSSLS remove the orientation toward linear differential equation which character analog methods.
  - **CSMP - III:**
    - Continuous System Modeling Program Version III
    - A type of CSSLS.
    - It is constructed from **three general type of statement:**
      1. Structural Statement
      2. Data Statements
      3. Control Statements

Structural , Data & Control Statements:

1. Structural Statements

- Structural statements define the **static configuration** and **relationships** within the system. They answer the question: **"How are the components of the system arranged and connected?"**
  - Structural statements which define the model, consist of FORTRAN-like programming language statements, and functional blocks of program-code (procedures) **designed for repeat operations** that frequently occur in a model definition.
  - Structural statements can make use of the **operations of addition, subtraction, multiplication, division and exponentiation**, using the same notation and rules as used in FORTRAN.
  - For e.g., the model includes the equation:

$$X = \frac{6Y}{W} + (Z - 2)^2$$

  - The following statement would be used as:

$$X = 6.0 \times \frac{Y}{W} + (Z - 2.0)^{2.0}$$

Structural , Data & Control Statements:

1. Structural Statements

- There are many **functional blocks** which in addition to providing operations specific simulation, duplicate many of the mathematical-function subprograms of FORTRAN.
- Among these are the **exponential functions, trigonometric functions** and functions for taking a maximum and minimum values.
- Below is the list of **eleven** of the functional blocks.

Functional Block	General Form	Description
INTEGRATOR	`Y = INTGRL(IC, X)`	Performs integration with an initial condition (`IC`). `Y(0) = IC`
LIMITER	`Y = LIMIT(P₁, P₂, X)`	Limits the value of input `X` between lower (`P₁`) and upper (`P₂`) bounds.
STEP FUNCTION	`Y = STEP(P)`	Generates a unit step function at time `P` .
EXPONENTIAL	`Y = EXP(P)`	Calculates the exponential function `e^P` .
NATURAL LOGARITHM	`Y = ALOG(X)`	Calculates the natural logarithm `ln(X)` .
TRIGONOMETRY SINE	`Y = SIN(X)`	Calculates the sine of `X` .
TRIGONOMETRY COSINE	`Y = COS(X)`	Calculates the cosine of `X` .
SQUARE ROOT	`Y = SQRT(X)`	Calculates the square root of `X` .
ABSOLUTE VALUE	`Y = ABS(X)`	Calculates the absolute value of `X` .
LARGEST VALUE	`Y = AMAX1(X₁, X₂, ..., Xₙ)`	Returns the maximum value from a list of inputs.
SMALLEST VALUE	`Y = AMIN1(X₁, X₂, ..., Xₙ)`	Returns the minimum value from a list of inputs.

Structural , Data & Control Statements:

2. Data Statements

Data statements are those which **assign numerical values** to various changing parameters, constants, and initial conditions of the data statements. They answer the question: **"What are the characteristics of the components and entities?"**

- **Purpose:** To define **system attributes, initial conditions, and input specifications. Example:** **INCON** can be used to set the initial values of the **integration-function block.**
- **Key Functions:**
  - **Parameters:** Defining **fixed values** that control system behavior (e.g., a processing time parameter for a machine).
  - **Attributes:** Defining properties carried by individual **entities** (e.g., a "job's" priority or arrival time).
  - **System Variables:** Defining dynamic values that track the **system's state** (e.g., the current number of entities in a queue, utilization of a resource).
- **Analogy:** This is equivalent to declaring **global variables** and **setting initial values**, or defining the fields/members of an object class.

Structural , Data & Control Statements:

3. Control Statements:

Control statements are the **most critical set of instructions**, which specify options in **assembly and execution** of the program, and the choice of output of the results of the calculations performed. They answer the question: **"What happens next, and when?"**

- **Purpose:** To manage the **flow of logic**, **allocate resources**, and **schedule events**.
- **Key Functions:**
  - **Time Advancement:** Commands like **WAIT** (for a specific duration), **HOLD** (a resource), or **SCHEDULE** (an event to occur at a future time).
  - **Resource Management:** Commands to acquire (**SEIZE or REQUEST**) and release (RELEASE) system resources.
  - **Conditional Logic and Looping:** Standard programming constructs (**IF-THEN-ELSE, WHILE**) used to model **decision-making processes** within the system (e.g., IF a queue is full, THEN block a new arrival).
  - **Flow Control:** Directing the movement of entities within the system (**TRANSFER, GO TO**).

Structural , Data & Control Statements:

Statement	Purpose	Example
`CONST`	Assigns constant values to variables for a specific run.	`CONST A = 0.5, XDOT = 1.25`
`PARAM`	Assigns a **series of values** to a parameter (used for parameter sweeps).	`PARAM D = (0.25, 0.50, 0.75, 1.0)`
`PRINT`	Outputs results in tabular form.	`PRINT X, XDOT, X2DOT`
`PRTPLT`	Outputs results in graph (plot) form.	`PRTPLT X`
`TIMER`	Specifies simulation timing: integration step, final time, and output intervals.	See detailed TIMER parameters below
`TITLE`	Adds a heading to the printed output.	`TITLE AUTOMOBILE SUSPENSION SYSTEM`
`LABEL`	Adds a heading to the print-plotted output.	`LABEL DISPLACEMENT VERSUS TIME`
`END`	Marks the end of all control and model statements.	`END`
`STOP`	Terminates the program.	`STOP`

TIMER Statement Parameters

Item	Meaning
`DELT`	Integration time step
`FINTIM`	Final simulation time
`PRDEL`	Time interval to print results
`OUTDEL`	Time interval to plot/graph results

Example of Timer Usage:

TIMER DELT = 0.005, FINTIM = 1.5,

PRDEL = 0.05, OUTDEL = 0.05

Structural , Data & Control Statements:

Example: Automobile Suspension System

Below is a complete example of modeling a simple **automobile suspension system** using the provided control statements.

```
TITLE AUTOMOBILE SUSPENSION SYSTEM
PARAM D = (5.656, 16.986, 39.592, 56.56, 113.12)
X2DOT = (1.0/M) * (K·F – K·X – D·XDOT)
XDOT = INTGRL (0.0, X2DOT)
X = INTGRL (0.0, XDOT)
CONST M = 2.0, F = 1.0, K = 400.0
TIMER DELT = 0.005, FINTIM = 1.5, PRDEL = 0.05, OUTDEL = 0.05
PRINT X, XDOT, X2DOT
PRTPLT X
LABEL DISPLACEMENT VERSUS TIME
END
STOP
```

Hybrid Simulation:

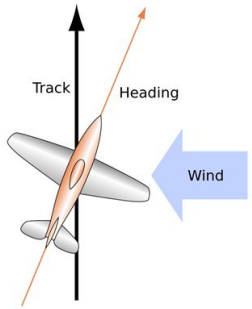
- An **analog and digital computers are combined** to provide a hybrid simulation depends upon the application.
- It uses of an analog or a digital computer for system simulation **depends on the model** (continuous or discrete).
- The system being simulated is an interconnection of continuous and discrete sub system, which can best be modeled by an **analog and digital computer being linked together**.
- **High speed converters** are needed to transform signals from **one form** of representation **to** the **other.**
- Hybrid simulation is generally reserved for the case in which **functionally distinct** analog and digital computers are **linked together** for the purpose of simulation example use of digital elements added to the **operational amplifiers** of an analog computers.
- **Hybrid Simulation = Discrete Event Simulation (DES) + Continuous Simulation (CS)**

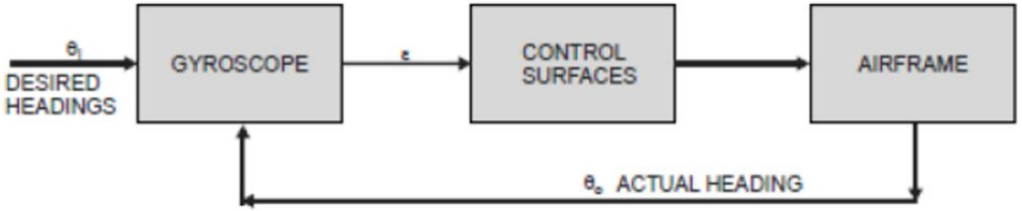
Feedback System:

- A feedback system is **one that compares its output (to a desired input) and takes corrective action to force the output to follow the input.**
- The term feedback is used to describe the phenomenon (situation).
- A home heating system controlled by a **thermostat** (a device that automatically regulates temperature) is a simple example of a feedback system.
- **Example :**
  - The automatic heater whose **purpose is to heat a room**, and the **output** of the system can be measured as a room temperature. Depending upon whether the **temperature is below or above the thermostat setting**, the heater will be turned on or off, so that information is being feedback from the **output to input**. In this case there are only two states either the heater is on or off.

Feedback System:

- Example: **Continuous System Control Mechanism using feedback is the Aircraft System.**
- Here the **input is a desired aircraft heading** and the **output is the actual heading**. The **gyroscope** of the autopilot is able to **detect the difference** between the two headings.
- A feedback is established by using the difference to operate the control surface. Since change of heading will then affect the signal being used to control the heading.
- The **difference between** the **desired heading and actual heading** is called the **error signal**, since it is a measure of the extent to which the system from the desired condition.





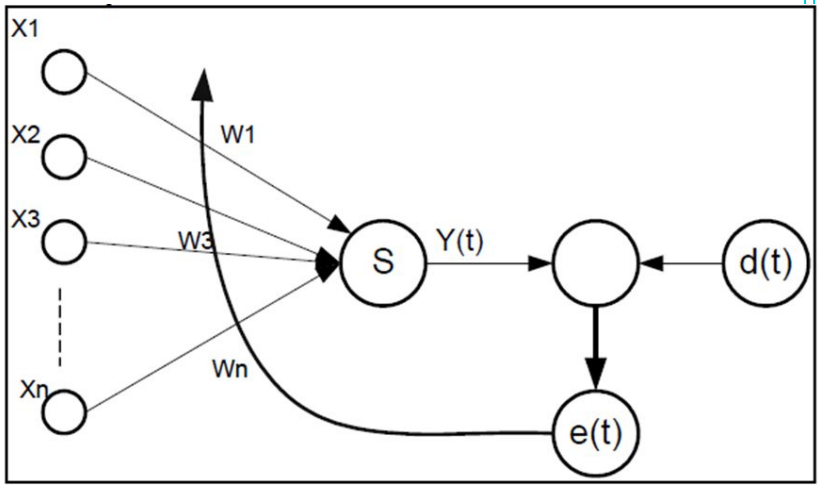
Feedback System:

- **Error Correction Mechanism**
- **Y(t)** – real output, **D(t)** – desired output
- Let error at that instant be;

  e(t) = D(t) – Y(t)

- **Y(t) is governed as;**

$$\sum_{i=0}^{n} w_i(t).x_i(t)$$



- e(t) is provided as feedback so that it adjusts the values of w(t)

The process is stopped if Y(t) = D(t)

Feedback System:

1. **Positive Feedback** – control variable and output are **inversely proportional**.

   - In a "positive feedback control system", the **output values are added together** by the controller as the feedback is "**in-phase**" with the **input**.
   - The effect of positive feedback is to "**increase**" the systems gain, i.e, the overall gain with positive **feedback applied will be greater than the gain without feedback**.

2. **Negative Feedback** - control variable is **proportional with output.**

   - In a "negative feedback control system", the **output values are subtracted from each other** as the feedback is "**out-of-phase**" with **the original input.**
   - The effect of negative feedback is to "**reduce**" the gain.

PYQs:

1. Discuss the limitations or challenges of using simulation languages for modeling complex systems. **2024 PU (8)**

References:

1. Jerry Banks, John S. Carson II, Barry L. Nelson, David M. Nicol – *Discrete-Event System Simulation* – Pearson (2013)
2. Averill M. Law – *Simulation Modeling and Analysis* – McGraw-Hill Education (2014)
3. Geoffrey Gordon – *System Simulation*

THANKS!

Do you have any questions?

theciceerguy@p_m.me

9841893035

ghimires.com.np