BACHELOR OF TECHNOLOGY YEAR 3

OBJECT ORIENTED ANALYSIS AND DESIGN

OBJECT-ORIENTED SYSTEMS & MODELLING

We know that the Object-Oriented Modelling (OOM) technique *visualizes things* in an application by using models organized around *objects*. Any software development approach goes through the following stages:-

- Analysis,
- Design, and
- Implementation

In object-oriented software engineering, the software developer *identifies* and *organizes* the application in terms of *object-oriented concepts*, prior to their final representation in any specific programming language or software tools.

PHASES IN OBJECT-ORIENTED SOFTWARE DEVELOPMENT

The major phases of software development using object–oriented methodology are:-

a) Object–Oriented Analysis

In this stage,

- The problem statement is formulated,
- User requirements are identified,
- A *model* is built based upon real–world objects.

The analysis produces *models* on how the desired system should *function* and how it must be *developed*.

The models do not include any implementation details so that it can be understood and examined by any non–technical application expert.

b) Object–Oriented Design

Object-oriented design includes two main stages, namely:-

i) System Design

A *complete architecture* of the desired system is designed. The system is conceived as a set of interacting subsystems that in turn is composed of a hierarchy of interacting objects, grouped into classes.

System design is done according to both the system analysis model and the proposed system architecture. Here, the emphasis is on the *objects* comprising the system rather than the *processes* in the system.

ii) Object Design

In this phase, a *design model* is developed based on both the *models* developed in the system analysis phase and the *architecture* designed in the system design phase.

All the classes required are identified and the designer decides whether:

- New classes are to be created from scratch,
- Any existing classes can be used in their original form, or
- New classes should be inherited from the existing classes.

The *associations* between the identified classes are established. The developer then designs the internal details of the classes and their associations, i.e., the *data structure* for each *attribute* and the *operations*.

c) Object–Oriented Implementation and Testing

In this stage, the *design model* developed in the object design is translated into *code* in an appropriate programming language or software tool.

The databases are created and the specific hardware requirements e.g. servers to host the applications are configured. Once the code is done, it is tested using specialized techniques to identify and remove the errors in the code.

OBJECT ORIENTED ANALYSIS TECHNIQUES

In the system analysis or object-oriented analysis phase of software development, the *system requirements* are determined, the *classes* are identified and the *relationships* among classes are

identified. The three analysis techniques that are used in conjunction with each other for object-oriented analysis are:-

a) Object Modelling

Object modelling develops the static structure of the software system in terms of objects. It identifies the objects, the classes into which the objects can be grouped into and the relationships between the objects.

It also identifies the main attributes and operations that characterize each class.

The process of object modelling can be visualized in the following steps:-

- Identify objects and group into classes
- Identify the relationships among classes
- Create user object model diagram
- Define user object attributes
- Define the operations that should be performed on the classes

b) Dynamic Modelling

After the static behavior of the system is analyzed, its behavior with respect to time and external changes needs to be examined. This is the main purpose of dynamic modelling.

Dynamic Modelling can be defined as "*a way of describing how an individual object responds to events, either internal events triggered by other objects, or external events triggered by the outside world*".

The process of dynamic modelling can be visualized in the following steps:-

- Identify states of each object
- Identify events and analyze the applicability of actions
- Construct dynamic model diagram, comprising of state transition diagrams
- Express each state in terms of object attributes
- Validate the state–transition diagrams

c) Functional Modelling

Functional Modelling is the final component of object-oriented analysis. It shows the processes that are performed within an object and how the data changes as it moves between methods.

It specifies the meaning of the *operations* of object modelling and the *actions* of dynamic modelling. The functional model corresponds to the *data flow diagrams.* The process of functional modelling can be visualized in the following steps:-

- Identify all the inputs and outputs
- Construct data flow diagrams showing functional dependencies
- State the purpose of each function
- Identify constraints
- Specify optimization criteria

DYNAMIC MODELLING

The *dynamic model* represents the *time–dependent aspects* of a system. It is concerned with the temporal changes in the states of the objects in a system. The main concepts are:-

- State: - which is the situation at a particular condition during the lifetime of an object.
- Transition: - a change in the state
- Event:- an occurrence that triggers transitions
- Action: - an uninterrupted and atomic computation that occurs due to some event, and concurrency of transitions.

A *state machine* models the *behavior* of an object as it passes through a number of states in its lifetime due to some *events* as well as the *actions* occurring due to the events.

A state machine is graphically represented through a *state transition diagram.*

STATES AND STATE TRANSITIONS

1. State

The *state* is a situation occurring for a finite time period in the lifetime of an object, in which it fulfils certain *conditions*, performs certain *activities*, or waits for certain *events* to occur.

In state transition diagrams, a state is represented by *rounded rectangles.*

Parts of a State

- **Name:** A string differentiates one state from another. A state may not have any name.
- **Entry/Exit Actions:** It denotes the activities performed on entering and on exiting the state.
- **Internal Transitions:** The changes within a state that do not cause a change in the state
- **Sub–states:** States within states

2. *Initial and Final States*

The default *starting* state of an object is called its *initial state.* The *final state* indicates the completion of execution of the state machine. The initial and the final states may not have the parts of a regular state except name. In state transition diagrams, the initial state is represented by *a filled black circle*. The final state is represented by a filled black circle encircled within another unfilled black circle as shown in the figure below.

3. *Transition*

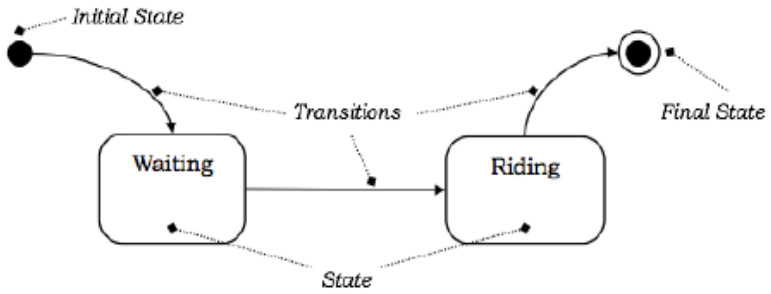
A *transition* denotes a *change* in the state of an object. If an object is in a certain state when an *event* occurs, the object may perform certain *activities* subject to specified conditions and change the state. In this case, a *state–transition* is said to have occurred. The transition gives the *relationship* between the first state and the new state. A transition is graphically represented by a *solid directed arc* from the source state to the destination state.

The five parts of a transition are:

- **Source State:** The state affected by the transition.
- **Event Trigger:** The occurrence due to which an object in the source state undergoes a transition if the guard condition is satisfied
- **Guard Condition:** A Boolean expression which if True, causes a transition on receiving the event trigger
- **Action:** An un-interruptible and atomic computation that occurs on the source object due to some event
- **Target State:** The destination state after completion of transition

Example

Suppose a person is taking a taxi from place X to place Y. The states of the person may be: Waiting (waiting for taxi), *Riding* (he has got a taxi and is travelling in it), and *Reached* (he has reached the destination). The following figure depicts the state transition.



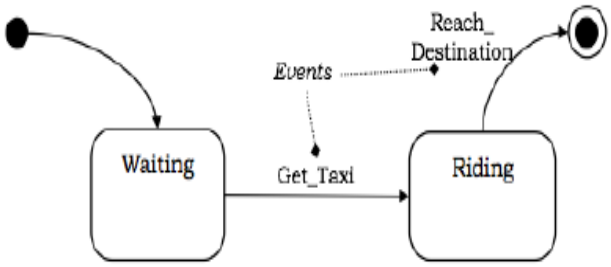
4. Events

Events are some *occurrences* that can *trigger* state transition of an object or a group of objects. Events have a location in time and space but do not have a time period associated with it. Events are generally associated with some actions.

Examples of events are *mouse click*, *key press*, etc.

Events that trigger transitions are written alongside the arc of transition in state diagrams.

Considering the example shown in the above figure, the transition from *Waiting* state to *Riding* state takes place when the person gets a taxi. Likewise, the final state is reached, when he reaches the destination. These two occurrences can be termed as events *Get_Taxi* and *Reach_Destination*. The following figure shows the events in a state machine.



Types of Events

i) External and Internal Events

External events are those events that pass from a user of the system to the objects within the system. For example, mouse click or key−press by the user are external events.

Internal events are those that pass from one object to another object within a system. For example, stack overflow, a divide error, etc.

ii) Event Classes

Event class indicates a group of events with common structure and behavior. As with classes of objects, event classes may also be organized in a hierarchical structure. Event classes may have attributes associated with them, time being an implicit attribute. For example, we can consider the events of departure of a flight of an airline, which we can group into the following class:-

Flight_Departs (Flight_No, From_City, To_City, Route)

5. Actions & Activities

i) Action

An *action* is an atomic operation that executes as a result of certain events. By atomic, we mean that actions are un-interruptible, i.e., if an action starts executing, it runs into completion without being interrupted by any event. An action may operate upon an object on which an event has been triggered or on other objects that are visible to this object. A set of *actions* comprise an *activity*.

ii) Activity

Activities are operations upon the states of an object .They are ongoing executions within a system that can be interrupted. Activities are shown in *activity diagrams* that portray the flow from one activity to another.

iii) Entry and Exit Actions

Entry action is the action that is executed on entering a state, irrespective of the transition that led into it.

Likewise, the action that is executed while leaving a state, irrespective of the transition that led it out is called an *exit action.*

6. Scenario

Scenario is a description of a specified sequence of actions. It depicts the behavior of objects undergoing a specific action series.

DIAGRAMS FOR DYNAMIC MODELLING

There are two primary diagrams that are used for dynamic modelling:-

INTERACTION DIAGRAMS

Interaction diagrams describe the dynamic behavior among different objects. It comprises of a set of *objects*, their *relationships*, and the *message* that the objects send and receive. Thus, an interaction models the behavior of a group of interrelated objects. There are two types of interaction diagrams:-

- **Sequence Diagram:** It represents the temporal ordering of messages in a tabular manner.
- **Collaboration Diagram:** It represents the structural organization of objects that send and receive messages through vertices and arcs.

STATE TRANSITION DIAGRAM

State transition diagrams or *state machines* describe the *dynamic behavior* of a single *object*. It shows the *sequences of states* that an object goes through in its lifetime, the *transitions of the states*, the *events and conditions* causing the transition and the *responses* due to the events.

CONCURRENCY OF EVENTS

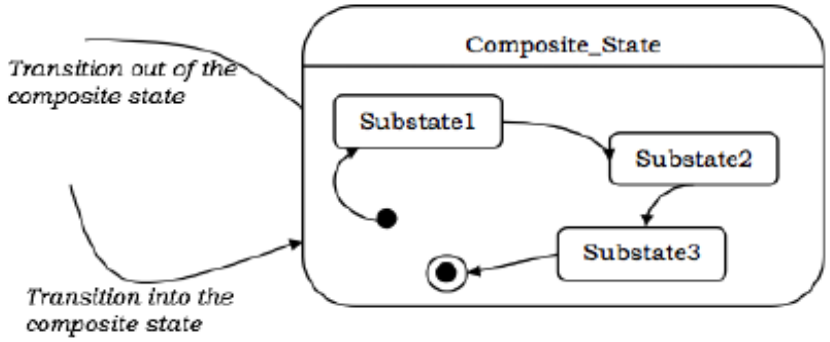
In a system, two types of concurrency may exist, namely:-

System Concurrency: - Here, concurrency is modelled in the system level. The overall system is modelled as the aggregation of state machines, where each state machine executes concurrently with others.

Concurrency within an Object: - Here, an object can issue concurrent events. An object may have states that are composed of sub-states, and concurrent events may occur in each of the sub-states.

Concepts related to concurrency within an object are as follows:-

- ***Simple and Composite States*:** A *simple state* has no sub-structure. A state that has simpler states nested inside it is called a *composite state*. A *sub-state* is a state that is nested inside another state. It is generally used to reduce the complexity of a state machine
- ***Sequential Sub-states*:-** The control of execution passes from one sub-state to another sub-state one after another in a sequential manner. There is at most one initial state and one final state in these state machines as shown below.



- ***Concurrent Sub-states*:-**The sub-states execute in parallel, or in other words, each state has concurrently executing state machines within it. Each of the state machines has its own initial and final states. If one concurrent sub-state reaches its final state before the other, control waits at its final state. When all the nested state machines reach their final states, the sub-states join back to a single flow as shown below:-