List of Practicals

Practical 1 : Study and implementation of class diagrams.

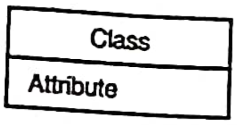
Solution :

In software engineering, a class diagram in the Unified Modeling Language (UML) is a type of static structure diagram that describes the structure of a system by showing the system's classes, their attributes, operations (or methods), and the relationships among objects.

Components

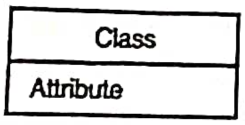
Class

- A class represents a relevant concept from the domain, a set of persons, objects, or ideas that are depicted in the IT system:
- Examples of classes are passengers, planes, or tickets.



Attribute

- An attribute of a class represents a characteristic of a class that is of interest for the user of the IT system :
- Characteristics of interest of a passenger, for example, are name and age.

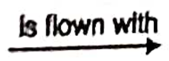


Generalization

- Generalization is a relationship between two classes: a general class and a special class :

Association

- An association represents a relationship between two classes :



- An association indicates that objects of one class have a relationship with objects of another class, in which this connection has a specifically defined meaning (for example, "is flown with").

Multiplicity

- A multiplicity allows for statements about the number of objects that are involved in an association :

Aggregation

- An aggregation is a special case of an association (see above) meaning "consists of" :
- The diamond documents this meaning; a caption is unnecessary.

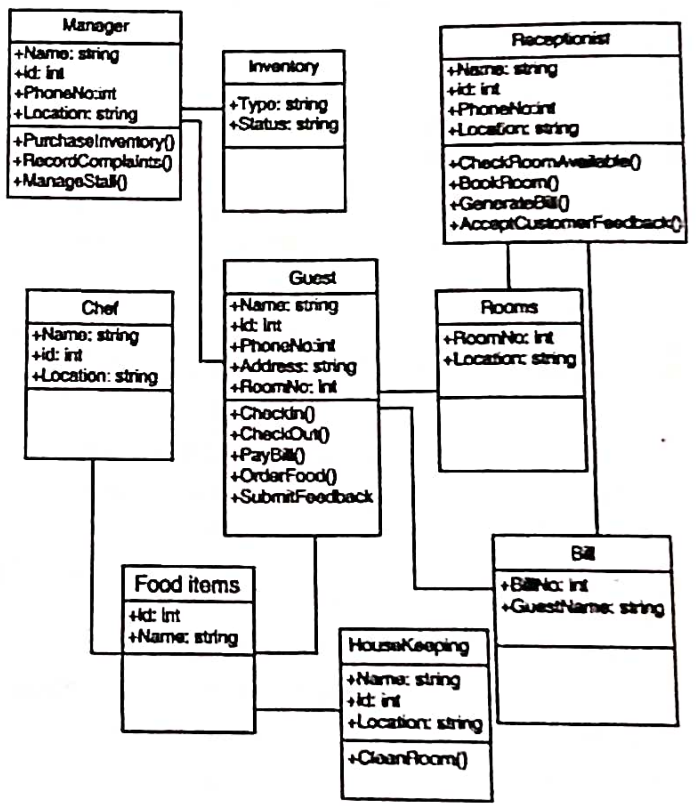


Fig. 1

Practical 2 : Study and implementation of Use Case Diagrams.

Solution :

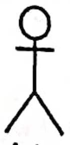
- A use case diagram is a graphic depiction of the interactions among the elements of a system.

- A use case is a methodology used in system analysis to identify, clarify, and organize system requirements.
- The actors, usually individuals involved with the system defined according to their roles.

Components

Actor :

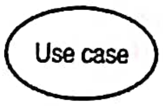
- You can picture an actor as a user of the IT system, for example Mr. Steel or Mrs. Smith from check-in.
- Because individual persons are irrelevant for the model, they are abstracted. So the actors are called "check-in employee" or "passenger":



- Actors represent roles that users take on when they use the IT system, e.g., the role of a check-in employee.
- One person can act in more than one role toward the IT system. It is important for the IT system in which role a person is acting. Therefore, it is necessary to log on to many IT systems in a certain role, for instance, as a normal user or as an administrator. In each case access to the appropriate functionalities (use cases) is granted.
- Actors themselves are not part of the IT system. However, as employees they can be part of the business system

Use Case

- Use cases describe the interactions that take place between actors and IT systems during the execution of business processes :



- A use case represents a part of the functionality of the IT system and enables the user (modeled as an actor) to access this functionality.
- Anything that users would like to do with the IT system has to be made available as a use case (or part of a use case). Functionalities that exist in the IT system, but that are not accessed by means of use cases, are *not* available to users.
- Even though the idea behind use cases is to describe interactions, flows of batch processing, which generally do not include interactions, can also be described as use cases. The actor of such a batch use case is then the one who initiates batch processing. For instance, *generating check-in statistics* would be a batch use case.

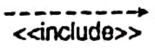
Association

- An association is a connection between an actor and a use case. An association indicates that an actor can carry out a use case. Several actors at one use case mean that each actor can carry out the use case on his or her own and not that the actors carry out the use case together:

- According to UML, association only means that an actor is involved in a use case. We use associations in a restricted manner.

Include Relationships

- An include relationship is a relationship between two use cases :



- It indicates that the use case to which the arrow points is included in the use case on the other side of the arrow.
- This makes it possible to reuse a use case in another use case. Fig. 1 shows an example of this relationship. In the flow of the use case, express check-in is a point at which the use case generating boarding pass is included.
- This means that at this point the entire process generating boarding pass is carried out. Include relationships can be viewed as a type of call to a subprogram:

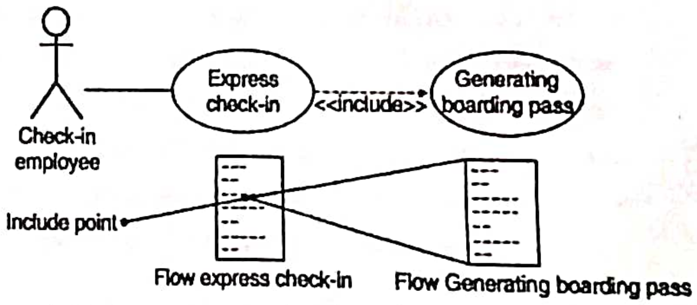


Fig. 2 : Include relationships between use cases

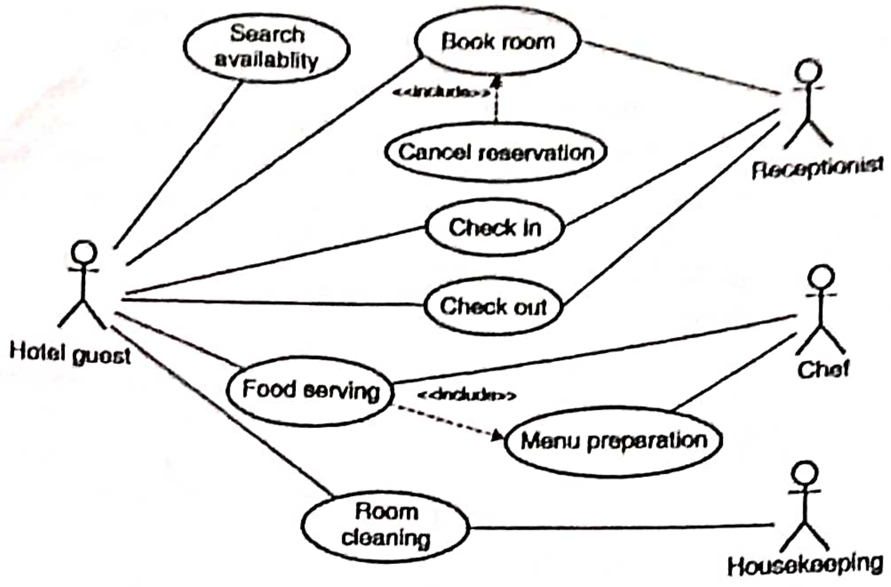


Fig. 3

Practical 3 : Study and implementation of Entity Relationship Diagrams.

Solution :

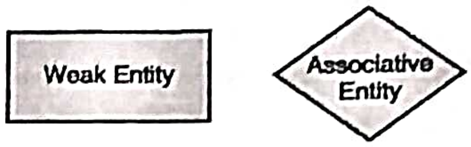
An entity relationship model, also called an entity-relationship (ER) diagram, is a graphical representation of entities and their relationships to each other, typically used in computing in regard to the organization of data within databases or information systems.

Components

Entity

A definable thing—such as a person, object, concept, or event—that can have data stored about it. Think of entities as nouns. Examples: a customer, student, car, or product. Typically shown as a rectangle.

- **Entity type:** A group of definable things, such as students or athletes, whereas the entity would be the specific student or athlete. Other examples: customers, cars or, products.
- **Entity set:** Same as an entity type, but defined at a particular point in time, such as students enrolled in a class on the first day. Other examples: Customers who purchased last month, cars currently registered in Florida. A related term is instance, in which the specific person or car would be an instance of the entity set.
- **Entity categories:** Entities are categorized as strong, weak, or associative. A strong entity can be defined solely by its own attributes, while a weak entity cannot. An associative entity associates entities (or elements) within an entity set.

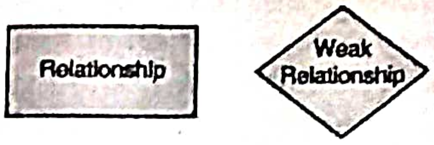


- **Entity keys:** Refers to an attribute that uniquely defines an entity in an entity set. Entity keys can be super, candidate, or primary.
  - **Super key:** A set of attributes (one or more) that together define an entity in an entity set.
  - **Candidate key:** A minimal super key, meaning it has the least possible number of attributes to still be a super key. An entity set may have more than one candidate key.
  - **Primary key:** A candidate key chosen by the database designer to uniquely identify the entity set.
  - **Foreign key:** Identifies the relationship between entities.

Relationship

- How entities act upon each other or are associated with each other. Think of relationships as verbs. For example, the named student might register for a course.

- The two entities would be the student and the course, and the relationship depicted is the act of enrolling, connecting the two entities in that way.
- Relationships are typically shown as diamonds or labels directly on the connecting lines.

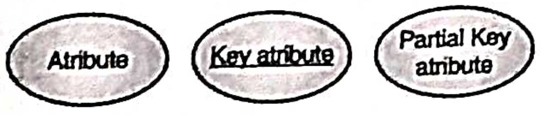


Recursive relationship

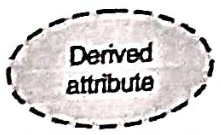
The same entity participates more than once in the relationship.

Attribute

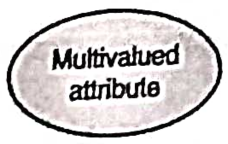
A property or characteristic of an entity. Often shown as an oval or circle.



- **Descriptive attribute:** A property or characteristic of a relationship (versus of an entity.)
- **Attribute categories :** Attributes are categorized as simple, composite, derived, as well as single-value or multi-value.
- **Simple :** Means the attribute value is atomic and can't be further divided, such as a phone number.
- **Composite :** Sub-attributes spring from an attribute.
- **Derived:** Attributed is calculated or otherwise derived from another attribute, such as age from a birthdate.

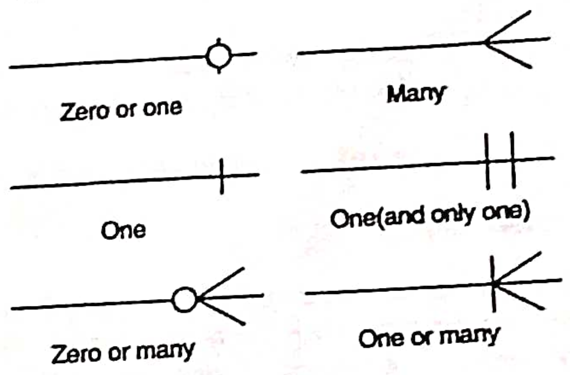


- **Multi-value :** More than one attribute value is denoted, such as multiple phone numbers for a person.
- **Single-value :** Just one attribute value. The types can be combined, such as: simple single-value attributes or composite multi-value attributes.



Cardinality

- Defines the numerical attributes of the relationship between two entities or entity sets. The three main cardinal relationships are one-to-one, one-to-many, and many-many.
- A one-to-one example would be one student associated with one mailing address. A one-to-many example (or many-to-one, depending on the relationship direction): One student registers for multiple courses, but all those courses have a single line back to that one student.
- Many-to-many example: Students as a group are associated with multiple faculty members, and faculty members in turn are associated with multiple students.



- **Cardinality views :** Cardinality can be shown as look-across or same-side, depending on where the symbols are shown.
- **Cardinality constraints :** The minimum or maximum numbers that apply to a relationship.

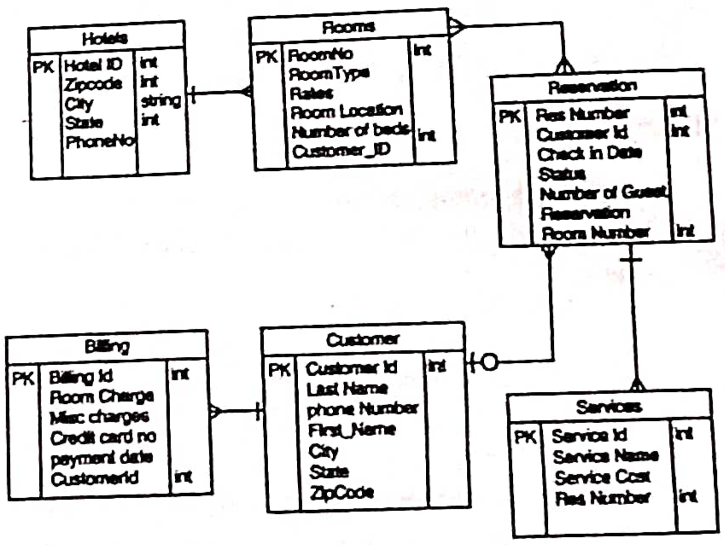


Fig. 4

Practical 4 : Study and implementation of Sequence Diagrams

Solution : The sequence diagram is a good diagram to use to document a system's requirements and to flush out a system's design. The reason the sequence diagram is so useful is because it shows the interaction logic between the objects in the system in the time order that the interactions take place.

Components

Comment

– The flow of a mutation event is documented with a combination of textual description and a sequence diagram:

– In comments, the flow logic is shown on the topmost level.

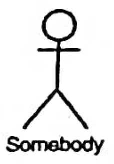
Delete Flight Number
Address Machine

...

Actor "Somebody"

– The actor "somebody" represents any actor from the use case diagram. Since the mutation event that is documented in a sequence diagram can be contained in several use cases, and since these use cases can have different actors, we use the actor somebody:

– This way, we do not have to decide on one, specific, actor.



Mutation Event

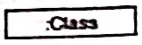
– A mutation event is an event that is sent from the use case, so normally from the user interface, to the IT system :

«M» Event/

– The goal of the event is to mutate information in the IT system, meaning to create, change, or delete something.

Object

– An object represents any object, meaning an undefined object of a class of the IT system:

Iteration 

– An iteration indicates that all objects to which a relationship exists receive the event, for example all the flights of a flight number:

*

Lifeline

– The lifeline of an object represents a life (over the course of time). The rectangle, meaning the "thick part" of the lifeline shows when the object is active:

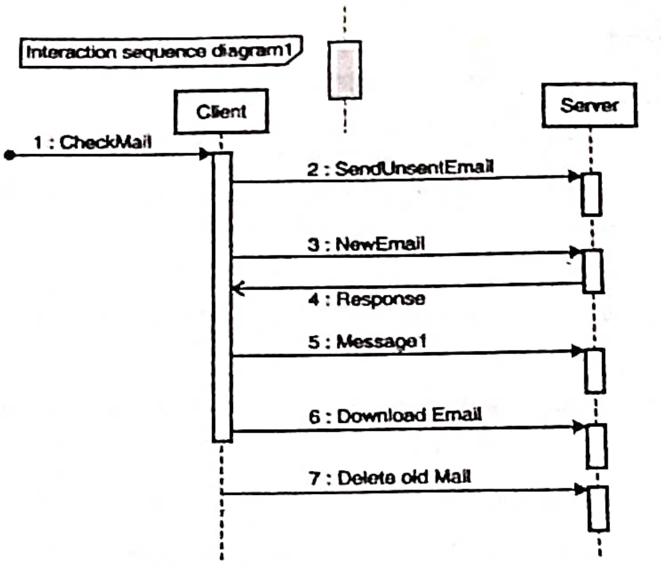


Fig. 5

Practical 5 : Study and implementation of State Transition Diagrams.

Solution : A state diagram is a diagram used in computer science to describe the behavior of a system considering all the possible states of an object when an event occurs. This behavior is represented and analyzed in a series of events that occur in one or more possible states.

Components

Initial State : The initial state represents the source of all objects:

●

It is not a normal state, because objects in this state do not yet exist.

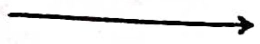
State

- The state of an object is always determined by its attributes and associations. States in state chart diagrams represent a *set* of those value combinations, in which an object *behaves the same* in response to events:

- Therefore, not every modification of an attribute leads to a new state.

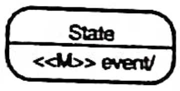
Transition

A transition represents the change from one state to another :



Internal Transition

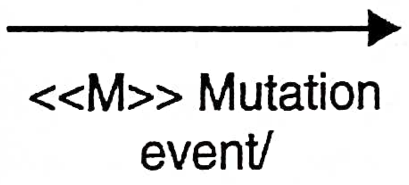
- An internal transition is a transition from one state to itself. This means that the object handles the event without changing its state :



- The events that initiate the internal transition are listed in the lower part of the state symbol. For instance, a frequent flyer card object in the state normal remains in the state normal when the event «M» add miles occurs.

Mutation Event

A mutation event is the initiator of a transition from one state to another, or for an internal transition, where the state remains the same :



Action : An action is the activity of an object that is initiated by an event:

«M» Event/Action

An action describes what the object does in response to the event. This description can be textual or formalized.

Guard Condition

- A guard condition is a condition that has to be met in order to enable the transition to which it belongs:
- Guard conditions can be used to document that a certain event, depending on the condition, can lead to different transitions.

[Guard Condition]

Final State

- The final state represents the end of an object's existence:

- A final state is not a real state, because objects in this state do not exist anymore.

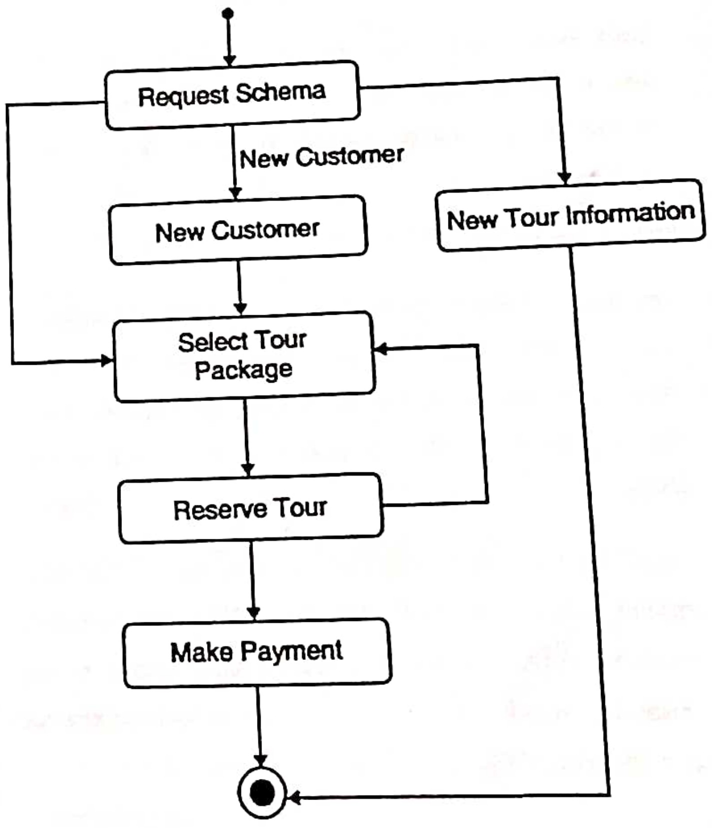


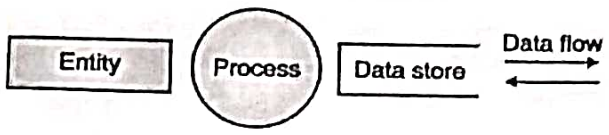
Fig. 6

Practical 6 : Study and implementation of Data Flow Diagrams.

Solution : A data flow diagram (or DFD) is a graphical representation of the flow of data through an information system. It shows how information is input to and output from the system, the sources and destinations of that information, and where that information is stored.

Components

DFD can represent Source, destination, storage and flow of data using the following set of components –



- **Entities :** Entities are source and destination of information data. Entities are represented by a rectangles with their respective names.
- **Process :** Activities and action taken on the data are represented by Circle or Round-edged rectangles.
- **Data Storage :** There are two variants of data storage - it can either be represented as a rectangle with absence of both smaller sides or as an open-sided rectangle with only one side missing.
- **Data Flow :** Movement of data is shown by pointed arrows. Data movement is shown from the base of arrow as its source towards head of the arrow as destination.

Levels of DFD

- **Level 0 :** Highest abstraction level DFD is known as Level 0 DFD, which depicts the entire information system as one diagram concealing all the underlying details. Level 0 DFDs are also known as context level DFDs.
- **Level 1 :** The Level 0 DFD is broken down into more specific, Level 1 DFD. Level 1 DFD depicts basic modules in the system and flow of data among various modules. Level 1 DFD also mentions basic processes and sources of information.

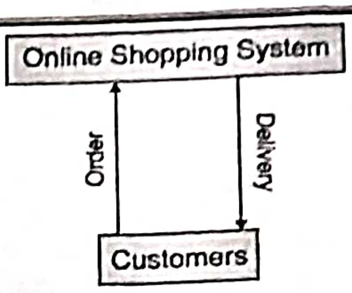


Fig. 7

- **Level 2 :** At this level, DFD shows how data flows inside the modules mentioned in Level 1.Higher level DFDs can be transformed into more specific lower level DFDs with deeper level of understanding unless the desired level of specification is achieved.

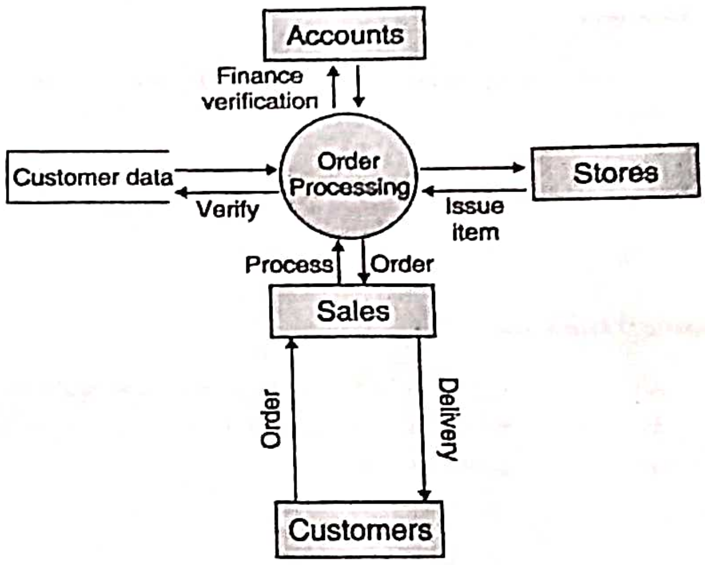


Fig. 8

Practical 7 : Study and implementation of Collaboration Diagrams.

Solution :

Collaboration diagram, also called a communication diagram or interaction diagram, is an illustration of the relationships and interactions among software objects in the Unified Modeling Language (UML).

Components

- **Object :** The interaction between objects takes place in a system. An object is depicted by a rectangle with the name of the object, preceded by a colon and underline.

- **Relation/Association:** Association among objects is linked by connecting them. The cardinality can be depicted by placing qualifiers on either ends.
- **Messages:** An arrow that commencing from one object to the destination object. This depicts the interaction between objects. The sequence or order of the interaction is depicted by the number

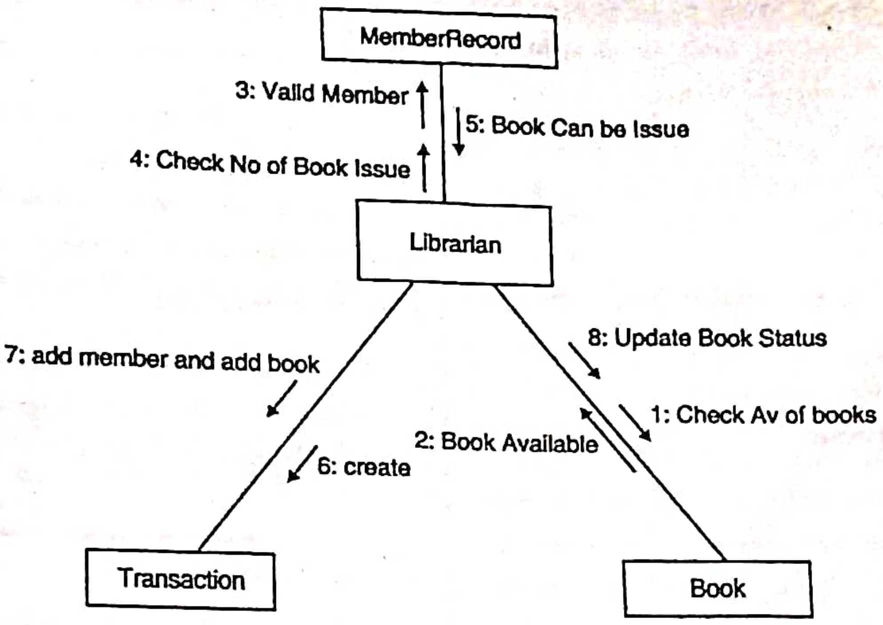


Fig. 9

Practical 8 : Study and implementation of Activity Diagrams.

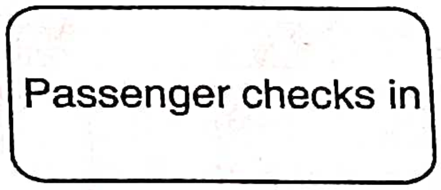
Solution :

- Activity diagram is another important diagram in UML to describe the dynamic aspects of the system.
- Activity diagram is basically a flowchart to represent the flow from one activity to another activity. The activity can be described as an operation of the system.

Components

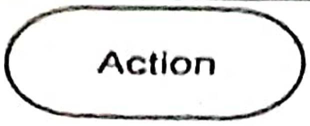
Activity

- An activity diagram illustrates one individual activity. In our context, an activity represents a business process . Fundamental elements of the activity are actions and control elements (decision, division, merge, initiation, end, etc.):
- Elements are connected by so-called "activity edges" and form the "control flow", which can also be casually called 'flow'. The execution of an activity can contain parallel flows. A border can surround the activity, meaning the entire activity diagram.



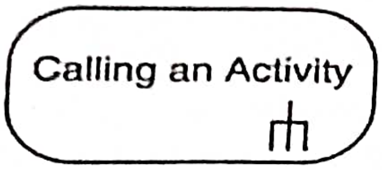
Action

- An action is an individual step within an activity, for example, a calculation step that is not deconstructed any further. That does not necessarily mean that the action cannot be subdivided in the real world, but in this diagram will not be refined any further:
- The action can possess input and output information The output of one action can be the input of a subsequent action within an activity. Specific actions are calling other actions, receiving an event, and sending signals.



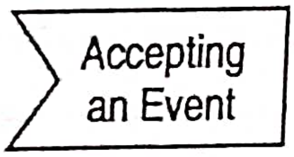
Calling an Activity (Action)

- With this symbol an activity can be called from within another activity. Calling, in itself, is an action; the outcome of the call is another activity.
- In this way, activities can be nested within each other and can be represented with different levels of detail.



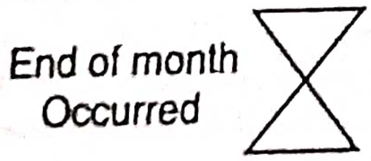
Accepting an Event (Action)

- This action waits for an event to occur. After the event is accepted, the flow that comes from this action (and is defined in the activity diagram) is executed. Accepting events is an important element for business processes in activity diagrams:
- Many business processes are initiated by events, for example, processing an order by the receipt of an order, or delivery by the receipt of a payment.



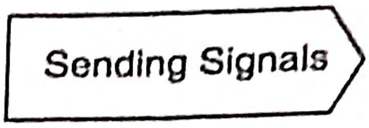
Accepting a Time Event (Action)

- At a definite point in time, this action starts a flow in the activity diagram. An hourglass symbol can be used to represent the acceptance of a time event:
- A typical example of a time event is triggering reminders after the deadline for payment has passed.



Sending Signals (Action)

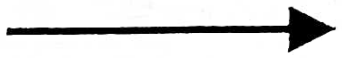
- Sending a signal means that a signal is being sent to an accepting activity :



- The accepting activity accepts the signal with the action "accepting an event" and can react accordingly, meaning according to the flow that originates from this node in the activity diagram.

Edge (Control Flow)

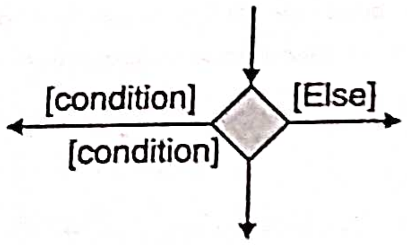
- Edges, represented by arrows, connect the individual components of activity diagrams and illustrate the control flow of the activity:



- Within the control flow an incoming arrow starts a single step of an activity after the step is completed the flow continues along the outgoing arrow. A name can be attached to an edge (close to the arrow).

Decision Node

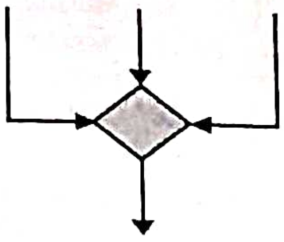
- The diamond below represents a conditional branch point or decision node. A decision node has one input and two or more outputs :



- Each output has a condition attached to it, which is written in brackets. If a condition is met, the flow proceeds along the appropriate output. An 'else' output can be defined along which the flow can proceed if no other condition is met.

Merge Node

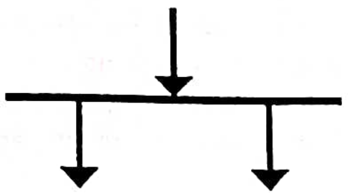
- The diamond below has several inputs and only one output :



- Its purpose is the merging of flows. The inputs are not synchronized; if a flow reaches such a node it proceeds at the output without waiting for the arrival of other flows.

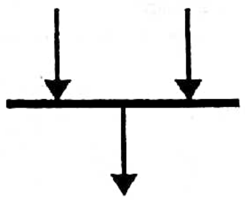
Fork

- For the branching of flows in two or more parallel flows we use a synchronization bar, which is depicted as a thick horizontal or vertical line:
- Branching allows parallel flows within activities. A fork has one input and two or more outputs.



Join

- For the consolidation of two or more parallel flows we also use a synchronization bar, which is depicted as a thick horizontal or vertical line:
- During consolidation synchronization takes place, meaning the flow proceeds only after all incoming flows have reached the consolidation point. Join has two or more inputs and one output.

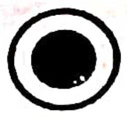


Initial Node : The initial node is the starting point of an activity. An activity can have more than one initial node; in this case several flows start at the beginning of an activity:

It is also possible that an activity has no initial node, but is initiated by an event (action: accepting an event).

Activity Final Node

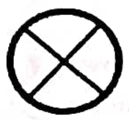
- The activity final node indicates that an activity is completed. An activity diagram can have more than one exit in the form of activity final nodes:



- If several parallel flows are present within an activity, all flows are stopped at the time the activity final node is reached.

Flow Final Node

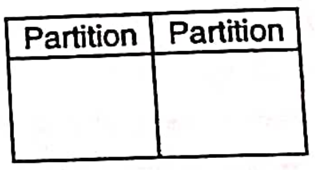
- A flow final node terminates a flow. Unlike the activity final node, which ends an entire activity, reaching a flow final node has no effect on other parallel flows that are being processed within the activity at the same point in time:



- In this way, parallel flows can be terminated individually and selectively.

Activity Partition

- The individual elements of an activity diagram can be divided into individual areas or 'partitions'. Various criteria can lead to the creation of these partitions: organization entities, cost centers, locations, etc.



- Individual steps of an activity will be assigned to these partitions. Each partition is set apart from its neighboring partition by a horizontal or vertical continuous line; from this stems the term *swim lanes*. Each partition receives a name. Partitions can be arranged in a two-dimensional manner; in this case the activity diagram is divided into individual cells like a grid.

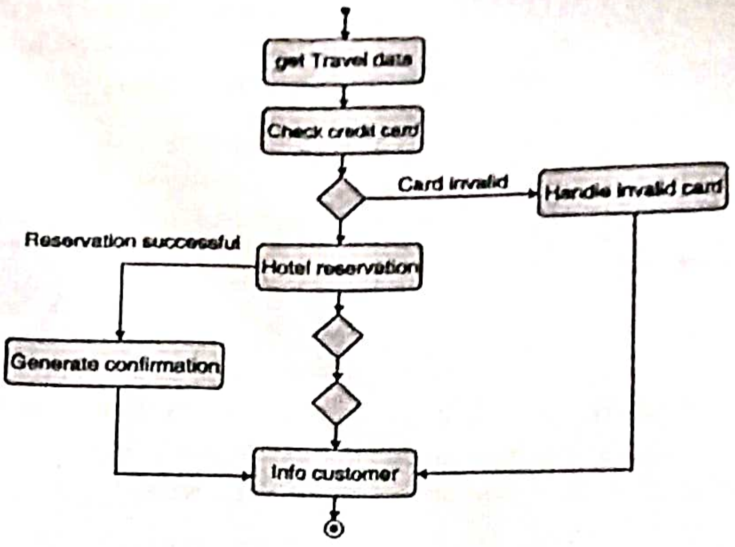


Fig. 10

Practical 9 : Study and implementation of Component Diagrams.

Solution :

UML Component Diagrams. Component diagram shows components, provided and required interfaces, ports, and relationships between them. This type of diagrams is used in Component-Based Development (CBD) to describe systems with Service-Oriented Architecture (SOA).

Components

Component - An entity required to execute a stereotype function. A component provides and consumes behavior through interfaces, as well as via other components. Think of components as a type of class. In UML 1.0, a component is modeled as a rectangular block with two smaller rectangles protruding from the side. In UML 2.0, a component is modeled as a rectangular block with a small image of the old component diagram shape.

- **Node :** Nodes are hardware or software objects, which are of a higher level than components. Boxes represent nodes in Lucidchart.
- **Interface :** Show input or materials that a component either receives or provides. Interfaces can be represented with textual notes or symbols such as the lollipop, socket, and ball-and-socket shapes.
- **Port :** Symbolized with a small square, ports specify a separate interaction point between the component and the environment.
- **Package :** Groups together multiple elements of the system. Packages are represented by file folders in Lucidchart. Just as file folders group together multiple sheets, packages can be drawn around several components.
- **Dependency :** Shows that one part of your system depends on another. Dependencies are represented by dashed lines linking one component (or element) to another.

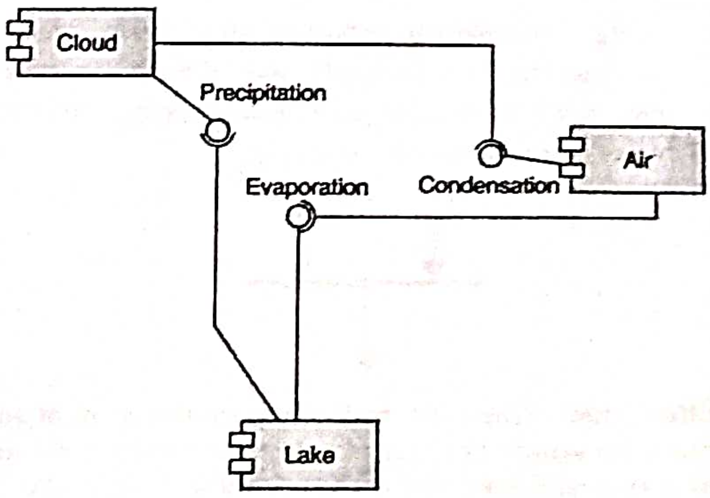


Fig. 11

Practical 10 : Study and Implementation of Deployment Diagrams.

Solution :

Deployment diagram is a structure diagram which shows architecture of the system as deployment (distribution) of software artifacts to deployment targets. Artifacts represent concrete elements in the physical world that are the result of a development process.

Components

- **Artifact :** A product developed by the software, symbolized by a rectangle with the name and the word "artifact" enclosed by double arrows.
- **Association :** A line that indicates a message or other type of communication between nodes.
- **Component :** A rectangle with two tabs that indicates a software element.
- **Dependency :** A dashed line that ends in a arrow, which indicates that one node or component is dependent on another.
- **Interface :** A circle that indicates a contractual relationship; those objects that realize the interface must complete some sort of obligation.
- **Node :** A hardware or software object, shown by a three-dimensional box.
- **Node as container :** A node that contains another node inside of it such as in the example below, where the nodes contain components.
- **Stereotype :** A device contained within the node, presented at the top of the node, with the name bracketed by double arrows.

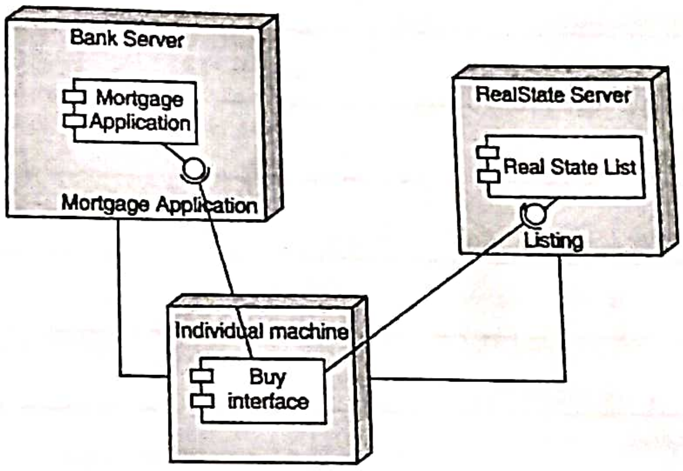


Fig. 12