



**JSPM, s
Imperial College of Engineering and Research
(ICOER)**

LABORATORY MANUAL

AY: 2023-24

**Laboratory Practice IV
410244(D) Object Oriented Modeling and Design**

BE Computer Engineering
Semester – I
Subject Code - 410247

TEACHING SCHEME
Practical: 2 Hrs / Week

CREDIT
01

EXAMINATION
TW: 50 Marks

Contents

Sr. No.	Problem Statement	Page No.
410244(D) Object Oriented Modeling and Design		
1	Draw state model for telephone line, with various activities.	3
2	Draw basic class diagram to identify and describe key concepts like classes, types in your system and their relationships.	9
3	Draw one or more Use Case diagrams for capturing and representing requirements of the system. Use case diagrams must include template showing description and steps of the Use Case for various scenarios	13
4	Draw activity diagrams to display either business flows or like flow charts.	19
5	Draw deployment diagrams to model the runtime architecture of your system.	22

410244(D) Object Oriented Modeling and Design

Group 1

EXPERIMENT No: 1

Title: Draw state model for telephone line, with various activities.

Relevant Theory:

State Chart Diagram Description

The UML state diagrams are directed graphs in which nodes denote states and connectors denote state transitions. In UML, states are represented as rounded rectangles labeled with state names. The transitions, represented as arrows, are labeled with the triggering events followed optionally by the list of executed actions. The initial transition originates from the solid circle and specifies the default state when the system first begins. Every state diagram should have such a transition, which should not be labeled, since it is not triggered by an event. The initial transition can have associated actions.

Purpose of State chart Diagrams

State chart diagram is one of the five UML diagrams used to model the dynamic nature of a system. They define different states of an object during its lifetime and these states are changed by events. State chart diagrams are useful to model the reactive systems. Reactive systems can be defined as a system that responds to external or internal events.

State chart diagram describes the flow of control from one state to another state. States are defined as a condition in which an object exists and it changes when some event is triggered. The most important purpose of State chart diagram is to model lifetime of an object from creation to termination.

State chart diagrams are also used for forward and reverse engineering of a system. However, the main purpose is to model the reactive system.

Following are the main purposes of using State chart diagrams

- To model the dynamic aspect of a system.
- To model the life time of a reactive system.
- To describe different states of an object during its life time.
- Define a state machine to model the states of an object.

How to Draw a State chart Diagram?

State chart diagram is used to describe the states of different objects in its life cycle. Emphasis is placed on the state changes upon some internal or external events. These states of objects are important to analyze and implement them accurately.

State chart diagrams are very important for describing the states. States can be identified as the condition of objects when a particular event occurs.

Before drawing a State chart diagram we should clarify the following points –

- Identify the important objects to be analyzed.
- Identify the states.
- Identify the events.

Where to Use State chart Diagrams?

State chart diagrams are used to model the dynamic aspect of a system like other four diagrams discussed in this tutorial. However, it has some distinguishing characteristics for modeling the dynamic nature.

State chart diagram defines the states of a component and these state changes are dynamic in nature. Its specific purpose is to define the state changes triggered by events. Events are internal or external factors influencing the system.

State chart diagrams are used to model the states and also the events operating on the system. When implementing a system, it is very important to clarify different states of an object during its life time and State chart diagrams are used for this purpose. When these states and events are identified, they are used to model it and these models are used during the implementation of the system.

If we look into the practical implementation of State chart diagram, then it is mainly used to analyze the object states influenced by events. This analysis is helpful to understand the system behavior during its execution.

The main usage can be described as –

- To model the object states of a system.
- To model the reactive system. Reactive system consists of reactive objects.
- To identify the events responsible for state changes.
- Forward and reverse engineering.

The main purposes of using State chart diagrams:

- To model dynamic aspect of a system.
- To model life time of a reactive system.
- To describe different states of an object during its life time.
- Define a state machine to model states of an object
- Identify important objects to be analyzed.
- Identify the states.

☐ Identify the events

State Notation:

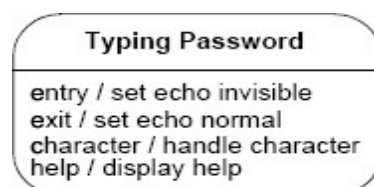


Figure 01- Building Blocks of a State chart Diagram

State

A state is any "distinct" stage that an object (system) passes through in its lifetime. An object remains in a given state for finite time until "something" happens, which makes it to move to another state. All such states can be broadly categorized into following three types:

- **Initial:** The state in which an object remains when created
- **Final:** The state from which an object does not move to any other state [optional]
- **Intermediate:** Any state, which is neither initial, nor final

As shown in figure-02, an initial state is represented by a circle filled with black. An intermediate state is depicted by a rectangle with rounded corners. A final state is represented by a unfilled circle with an inner black-filled circle.

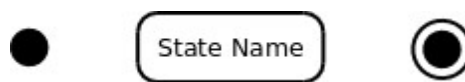


Figure 02- Representation of initial, intermediate, and final states of a state chart diagram

Intermediate states usually have two compartments, separated by a horizontal line, called the Name compartment and internal transitions compartment. They are described below:

- **Name compartment:** Contains the name of the state, which is a short, simple, descriptive string
- **Internal transitions compartment:** Contains a list of internal activities performed as long as the system is in this state

The internal activities are indicated using the following syntax: action-label / action-expression. Action labels could be any condition indicator. There are, however, four special action labels.

- **Entry:** Indicates activity performed when the system enters this state
- **Exit:** Indicates activity performed when the system exits this state
- **Do:** indicate any activity that is performed while the system remains in this state or until the action expression results in a completed computation
- **Include:** Indicates invocation of a sub-machine

Any other action label identifies the event (internal transition) as a result of which the corresponding action is triggered. Internal transition is almost similar to self-transition, except that the former doesn't result in execution of entry and exit actions. That is, the system doesn't exit or re-enter that state.

shows the typical state in a state chart diagram. States could again be either simple or composite (a state containing other states). Here, however, we will deal only with simple states.

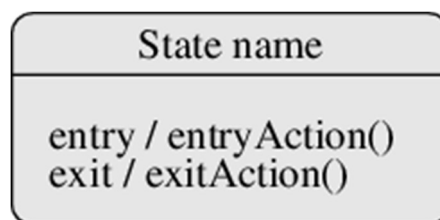


Figure 03. syntax for representing a typical (intermediate) state

Transition

Transition is movement from one state to another state in response to an external stimulus (or any internal event). A transition is represented by a solid arrow from the current state to the next state. It is labeled by: event [guard-condition]/[action-expression], where

- **Event** is the what is causing the concerned transition (mandatory) -- Written in past tense
- **Guard-condition** is (are) precondition(s), which must be true for the transition to happen [optional]
- **Action-expression** indicate action(s) to be performed as a result of the transition [optional]

It may be noted that if a transition is triggered with one or more guard-condition(s), which evaluate to false, the system will continue to stay in the present state. Also, not all transitions do result in a state change. For example, if a queue is full, any further attempt to append will fail until the delete method is invoked at least once. Thus, state of the queue doesn't change in this duration.

Action

As mentioned in , actions represents behavior of the system. While the system is performing any action for the current event, it doesn't accept or process any new event. The order in which different actions are executed, is given below:

- 1.Exit actions of the present state
- 2.Actions specified for the transition
- 3.Entry actions of the next state

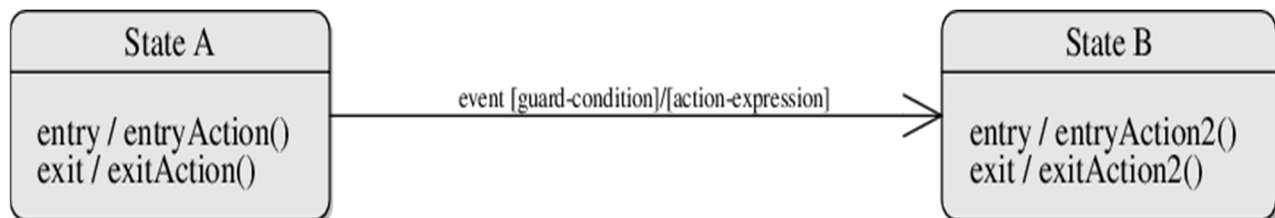
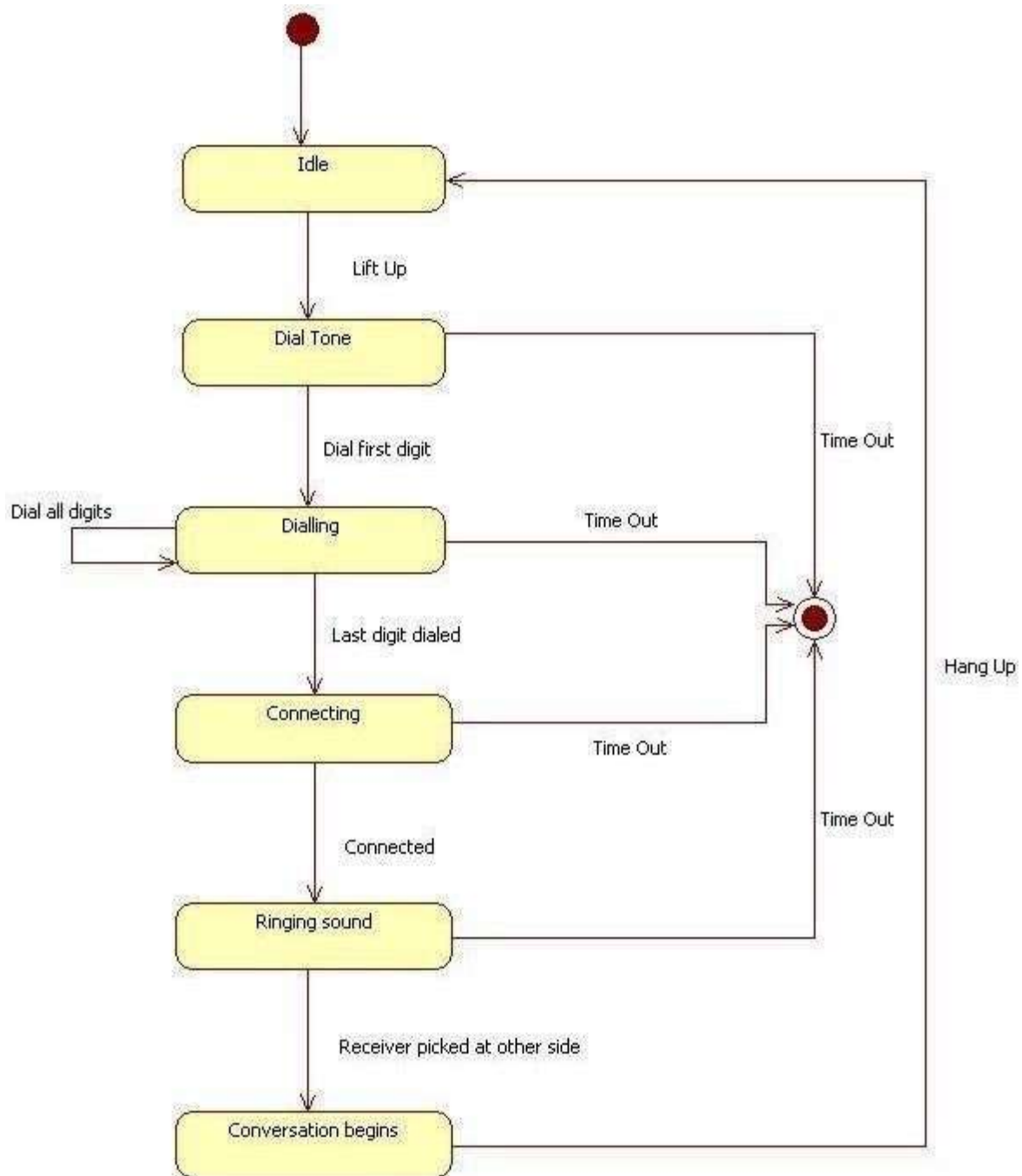


Fig-04: A state chart diagram showing transition from stateA to B

State chart diagram for telephone line :



Conclusion:

In this experiment, we studied the state chart diagram for telephone line, with various activities.

FAQ'S

1. What should be identified before drawing a state chart diagram?
2. What is the purpose of state chart diagram?
3. What are the limitations of state diagram?
4. What is state Chart Diagram?
5. What are the elements included in state chart diagrams?
6. What are the different states in the state chart diagrams?

EXPERIMENT No: 2

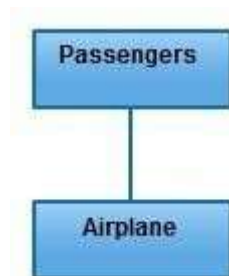
Title:-Draw basic class diagram to identify and describe key concepts like classes, types in your system and their relationships.

Relevant Theory:

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.

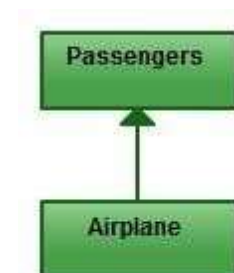
- **Class Diagram Relationships:**
 1. Association
 2. Directed Association
 3. Reflexive Association
 4. Multiplicity
 5. Aggregation
 6. Composition
 7. Inheritance/Generalization
 8. Realization

- **Association**



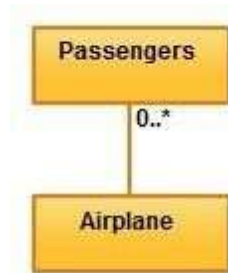
Association is a broad term that encompasses just about any logical connection or relationship between classes. For example, passenger and airline may be linked as above.

- **Directed Association**



Directed Association refers to a directional relationship represented by a line with an arrowhead. The arrowhead depicts a container-contained directional flow.

- **Multiplicity**



Multiplicity is the active logical association when the cardinality of a class in relation to another is being depicted. For example, one fleet may include multiple airplanes, while one commercial airplane may contain zero to many passengers. The notation 0..* in the diagram means “zero to many”.

- **Aggregation**



Aggregation refers to the formation of a particular class as a result of one class being aggregated or built as a collection. For example, the class “library” is made up of one or more books, among other materials. In aggregation, the contained classes are not strongly dependent on the lifecycle of the container. In the same example, books will remain so even when the library is dissolved. To show aggregation in a diagram, draw a line from the parent class to the child class with a diamond shape near the parent class. To show aggregation in a diagram, draw a line from the parent class to the child class with a diamond shape near the parent class.

- **Composition**

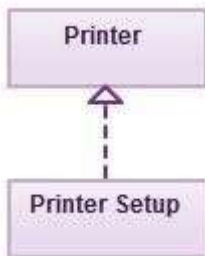


The composition relationship is very similar to the aggregation relationship. with the only difference being its key purpose of emphasizing the dependence of the contained class to the life cycle of the container class. That is, the contained class will be obliterated when the container class is destroyed. For example, a shoulder bag’s side pocket will also cease to exist once the shoulder bag is destroyed. To show a composition relationship in a UML diagram, use a directional line connecting the two classes, with a filled diamond shape adjacent to the container class and the directional arrow to the contained class.

- **Inheritance / Generalization**

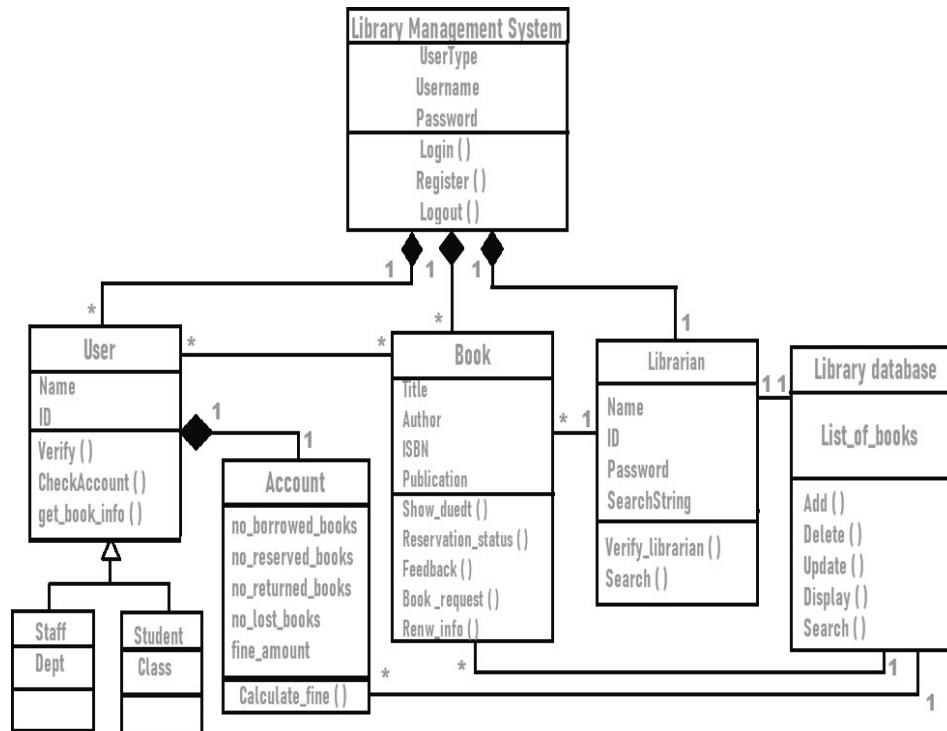


Inheritance refers to a type of relationship wherein one associated class is a child of another by virtue of assuming the same functionalities of the parent class. In other words, the child class is a specific type of the parent class. To show inheritance in a UML diagram, a solid line from the child class to the parent class is drawn using an unfilled arrowhead.



- **Realization**

Realization denotes the implementation of the functionality defined in one class by another class. To show the relationship in UML, a broken line with an unfilled solid arrowhead is drawn from the class that defines the functionality of the class that implements the function. In the example, the printing preferences that are set using the printer setup interface are being implemented by the printer.



CLASS DIAGRAM FOR LIBRARY MANAGEMENT SYSTEM

Conclusion:

Thus we have studied how to use class diagrams to identify and describe key concepts like classes, types in system and their relationships.

FAQ's

1. What is the common use of class diagram?
2. What are the 4 things needed to complete a class diagram?
3. Where can you apply class diagram?
4. What is the drawback of class diagram?
5. What is class diagram?
6. What are different relationships in class diagram?
7. What are the 3 main elements of a class diagram?
8. What is the purpose of a class diagram?

EXPERIMENT NO:3

Title: Draw one or more Use Case diagrams for capturing and representing requirements of the system. Use case diagrams must include template showing description and steps of the Use Case for various scenarios.

Relevant Theory:

Guideline: To Find Use Cases

Use cases are defined to satisfy the goals of the primary actors. Hence, the basic procedure is:

Step 1: Choose the system boundary. Is it just a software application, the hardware and application as a unit, that plus a person using it, or an entire organization?

Step 2: Identify the primary actors those that have goals fulfilled through using services of the system. The following questions help identify others that may be missed:

Who starts and stops the system?	Who does system administration?
Who does user and security management?	Is "time" an actor because the system does something in response to a time event?
Is there a monitoring process that restarts the system if it fails?	Who evaluates system activity or performance?
How are software updates handled? Push or pull update?	Who evaluates logs? Are they remotely retrieved?
In addition to human primary actors, are there any	Who gets notified when there are
Who starts and stops the system?	Who does system administration?
External software or robotic systems that call upon services of the system?	Errors or failures?

Step 3: Identify the goals for each primary actor.

For example, use this table to prepare actor goal list

Actor	Goal
Participant	Play level 1
	Play level 2
	Submit answers
...	...

Step 4: Define Use Cases

In general, define one use case for each user goal. Name the use case similar to the user goal. Start the name of use cases with a verb.

A common exception to one use case per goal is to collapse CRUD (create, retrieve, update, delete) separate goals into one CRUD use case, idiomatically called Manage <X>. For example, the goals "edit user," "delete user," and so forth are all satisfied by the Manage Users use case.

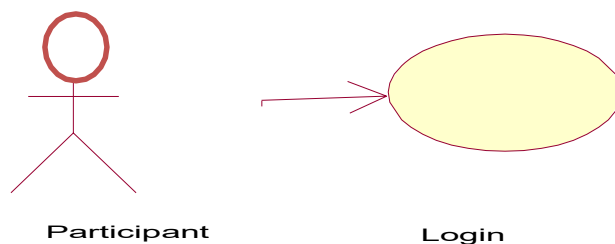
UML Notation:

Actor : An Actor, as mentioned, is a user of the system, and is depicted using a stick figure. The role of the user is written beneath the icon. Actors are not limited to humans. If a system communicates with another application, and expects input or delivers output, then that application can also be considered an actor.

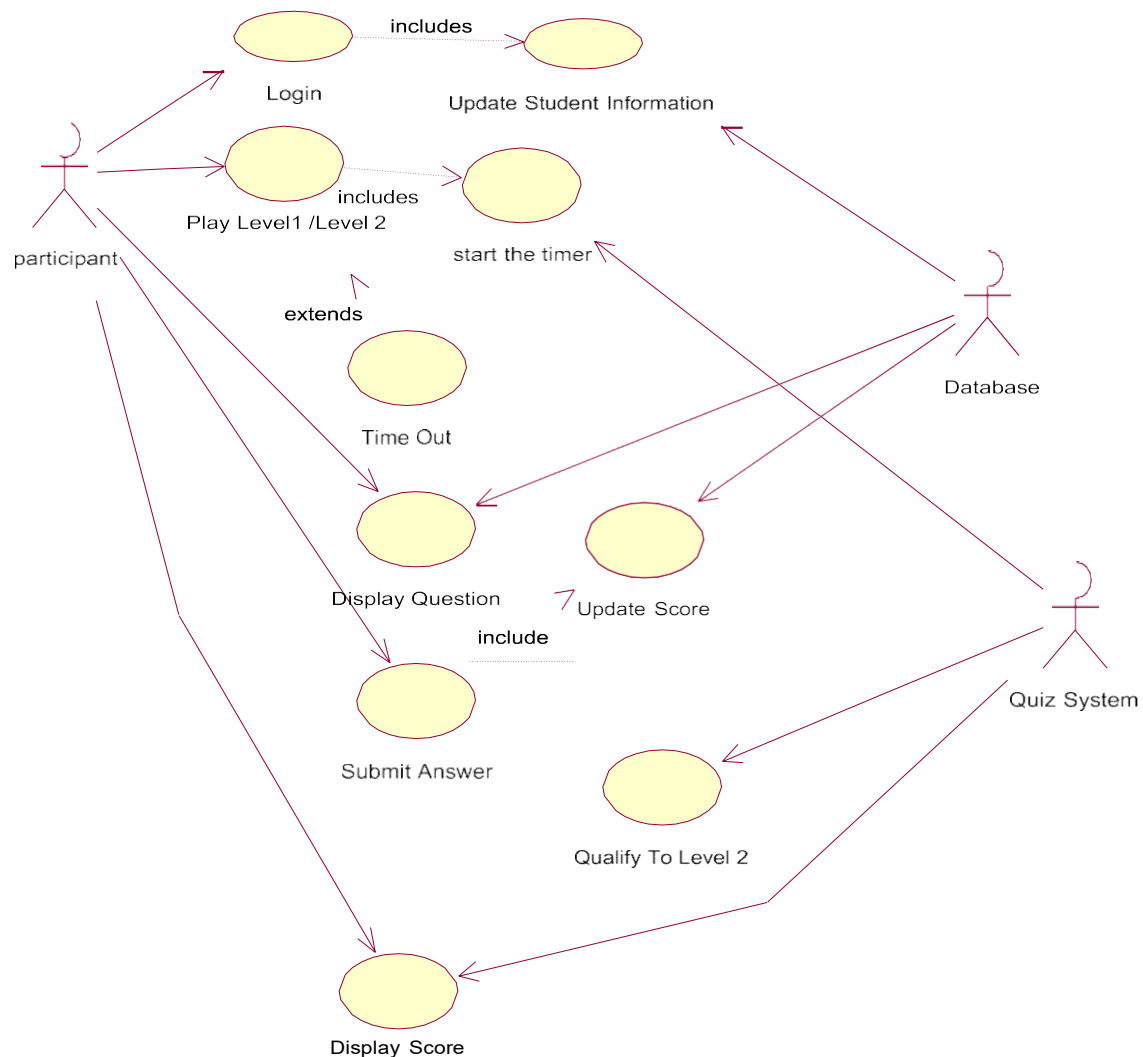
Use Case: A Use Case is functionality provided by the system, typically described as verb+object (e.g. Register Car, Delete User). Use Cases are depicted with an ellipse. The name of the use case is written within the ellipse.

Association: Associations are used to link Actors with Use Cases, and indicate that an Actor participates in the Use Case in some form. Associations are depicted by a line connecting the Actor and the Use Case.

Actor Association Use case



Use-case Diagram: Quiz System



- **Use cases**

The system will consist of Login screen to authenticate the participants whose information is updated in the database. On starting the quiz, timer is started to maintain the timings. In Level 1 /Level 2 Questions with four options are displayed sequentially .User select the answer and move to the next question. Finally he/she selects the Submit answers which updates the marks and displays the score to the participants. If he is playing Level 1 and qualified for level 2, then next level questions are displayed otherwise Not Qualified Message is displayed.

Functional Requirements

Use Case: Login Brief Description

The use case describes how a Participant logs into the Quiz System

Use Case Section	Comment
Use Case Name	Login
Scope	Quiz System
Level	"user-goal"

Use Case Section	Comment
Primary Actor	Participant
Stakeholders and Interest list	- Participant: logs into the Quiz System ...
Preconditions	None
Success Guarantee/ Post condition	If the use case was successful, the actor is now logged into the system. If not, the system State is unchanged.
Main Success Scenario	1. The System requests the actor to enter his/her name and password 2. The actor enters his/her name and password 3. The System validates the entered name and password and logs the actor into the System
Extensions	3a.If the pwd is wrong, user is allowed for 3 attempts
Special Requirements	Timer
Technology and Data Variations List	-
Frequency of Occurrence	Could Be nearly Continuous
Miscellaneous	If the connection is terminated before the form is submitted, the fields are cleared and the Server is returned to the wait state. The Administrator can make the system not to get updated by others.

Non-functional requirements

Non-functional requirements are often called qualities of a system. Other terms for non-functional requirements are "constraints", "quality attributes", "quality goals", "quality of service requirements" and "non-behavioral requirements".

Non-functional requirements, can be divided into two main categories:

- Execution qualities, such as security and usability, which are observable at run time.
- Evolution qualities, such as testability, maintainability, extensibility and scalability, which are embodied in the static structure of the software system

Functionality: Multiple users must be able to perform their work concurrently. If the participant has completed 30 Minutes allotted for him/her, he or she should be notified with the message “your time slot is over”.

Usability: The desktop user-interface shall be Windows 95/98/2000/xp compliant.

Reliability: The System should function properly for allotted time slot and produces score card with no more than 10% down time.

Performance

- The System shall support up to 100 simultaneous users against the central database at any given time and up to 100 simultaneous users against the local servers at any one time.
- The System must be able to complete 80% of all transactions within 2 minutes.

Security

- The System should secure so that only registered participants can take part in Quiz.
- Once the participant had submitted an answer, he/she can't change the answer later.

Design Constraints:

The system shall provide a window-based desktop interface

Conclusion:

In this experiment, we studied the Use Case diagrams for capturing and representing requirements of the system comprising the use case diagram and use case scenarios.

FAQ's

1. What are the 4 main components of a use case diagram?
2. What are the 2 types of use case diagram?
3. What is the main use of use case diagram?
4. What should not be shown on a use case diagram?
5. What is the most important rule for use case diagrams?
6. What is Use Case Diagram

EXPERIMENT No: 4

Title: Draw activity diagrams to display either business flows or like flow charts.

Relevant Theory:








What is an Activity diagram?

A UML activity diagram helps to visualize a certain use case at a more detailed level. It is a behavioral diagram that illustrates the flow of activities through a system.

UML activity diagrams can also be used to depict a flow of events in a business process. They can be used to examine business processes in order to identify their flow and requirements.

Activity Diagram Symbols

UML has specified a set of symbols and rules for drawing activity diagrams. Following are the commonly used activity diagram symbols with explanations.

Symbol	Name	Use
	Start/ Initial Node	Used to represent the starting point or the initial state of an activity
	Activity / Action State	Used to represent the activities of the process
	Action	Used to represent the executable sub-areas of an activity
	Control Edge	Used to represent the flow of control from one action to the other
	Object Edge	Used to represent the path of objects moving through the activity
	Activity Final Node	Used to mark the end of all control flows within the activity
	Flow Final Node	Used to mark the end of a single control flow



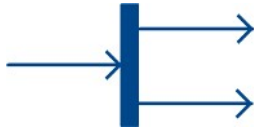
Decision
Node

Used to represent a conditional branch point with one input and multiple outputs



Merge
Node

Used to represent the merging of flows. It has several inputs, but one output.



Fork

Used to represent a flow that may branch into two or more parallel flows



Merge

Used to represent two inputs that merge into one output



Signal
Sending

Used to represent the action of sending a signal to an accepting activity



Signal
Receipt

Used to represent that the signal is received



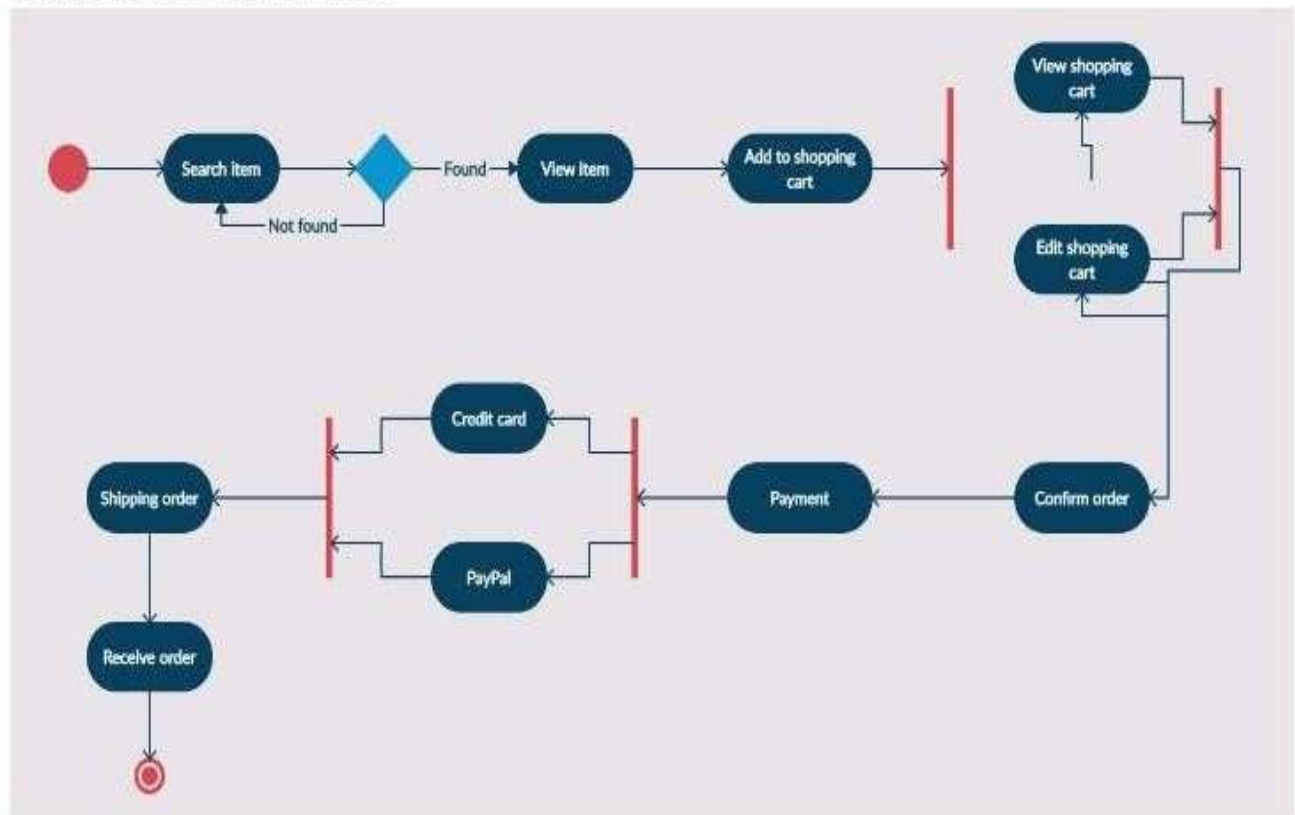
Note/
Comment

Used to add relevant comments to elements

Activity Diagram Examples

Activity Diagram for Online Shopping System

ONLINE SHOPPING SYSTEM for ABC Co.



Conclusion: Thus we have studied how to use activity diagram to find out behavior of any system. System.

FAQ

1. What are the limitations of activity diagram?
2. What can activity diagrams be used for?
3. Can an activity diagram have multiple endings?
4. Can activity diagram have two final states?
5. What are the important elements of activity diagram?
6. What are the four purposes of activity diagrams?
7. Can activity diagram have two start points?
8. What is activity diagram based on?
9. What are the 6 basic elements of an activity diagram?

EXPERIMENT No: 5

Title: Draw deployment diagrams to model the runtime architecture of your system.

Relevant Theory:

What is Deployment Diagram:-

A deployment diagram is a UML diagram type that shows the execution architecture of a system, including nodes such as hardware or software execution environments, and the middleware connecting them. Deployment diagrams are typically used to visualize the physical hardware and software of a system. Using it you can understand how the system will be physically deployed on the hardware.

Purpose of Deployment Diagrams:-

- They show the structure of the run-time system
- They capture the hardware that will be used to implement the system and the links between different items of hardware.
- They model physical hardware elements and the communication paths between them
- They can be used to plan the architecture of a system.
- They are also useful for Document the deployment of software components or nodes

Deployment Diagram at a Glance:-

Deployment diagrams are important for visualizing, specifying, and documenting embedded, client/server, and distributed systems and also for managing executable systems through forward and reverse engineering.

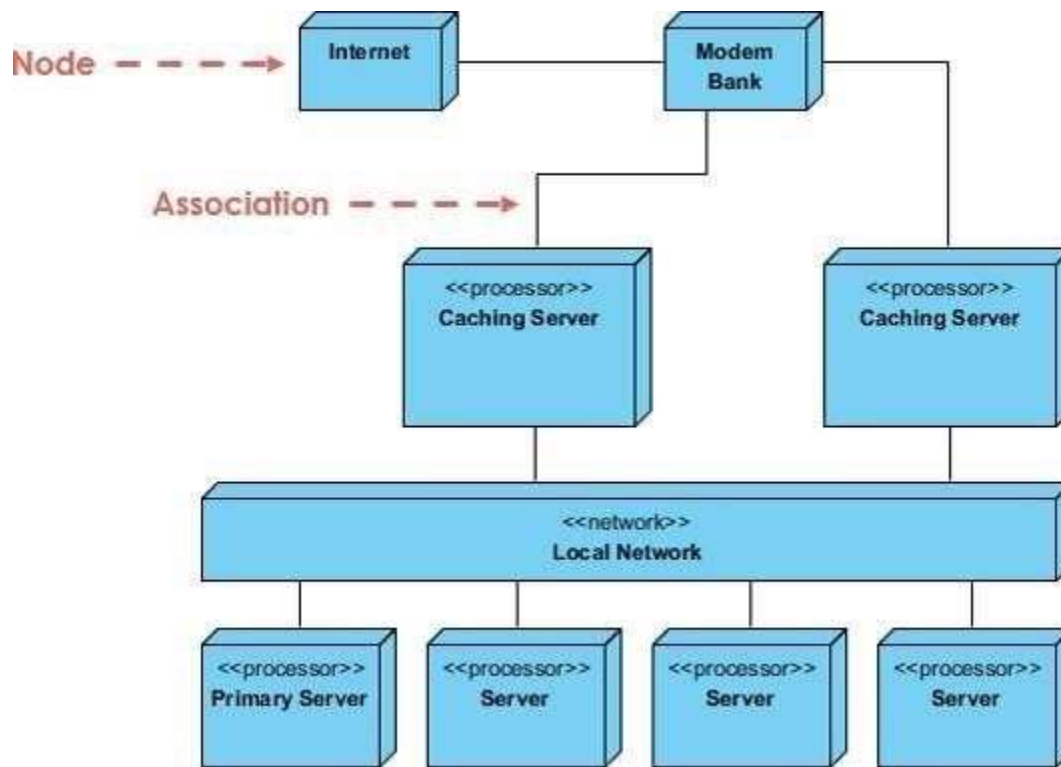
A deployment diagram is just a special kind of class diagram, which focuses on a system's nodes. Graphically, a deployment diagram is a collection of vertices and arcs. Deployment diagrams commonly contain:

Nodes

- 3-D box represents a node, either software or hardware
- HW node can be signified with <<stereotype>>
- Connections between nodes are represented with a line, with optional <<stereotype>>
- Nodes can reside within a node

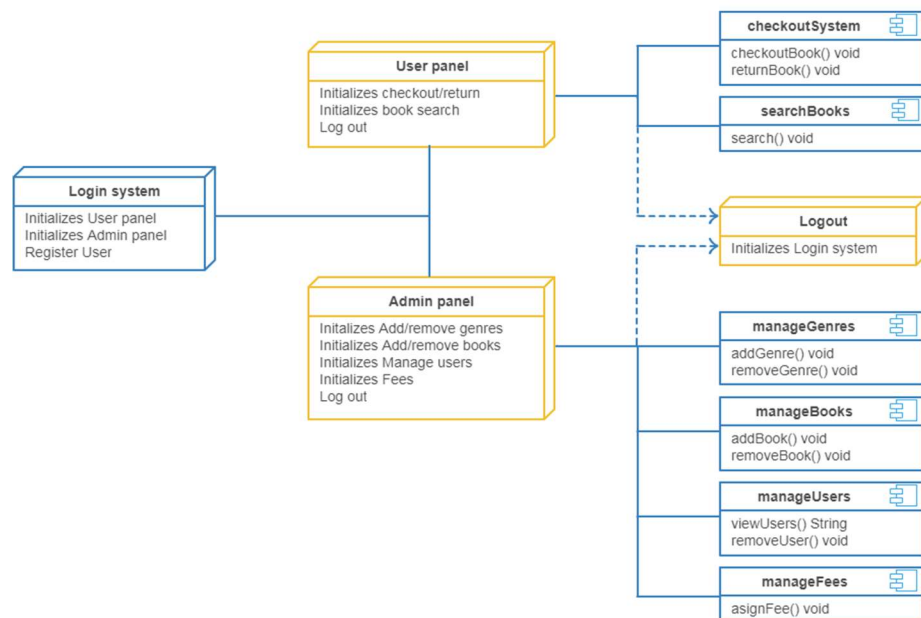
Other Notations

- Dependency
- Association relationships.
- May also contain notes and constraints



Deployment diagram

Deployment Diagram for Library Management System



Conclusion:

Thus we have studied what is deployment diagram and how to use it and visualize the system.

FAQ's

1. What is the main purpose of deployment diagram?
2. What are the two types of deployment diagram?
3. What does a deployment diagram consists of?
4. What are the common notations for deployment diagram?
5. Is deployment diagram static?
6. Where can you apply deployment diagram?