3. SYSTEM DESIGN

System Design is the process or art of defining the architecture, components, modules, interfaces, and data for a system to satisfy specified requirements. One could see it as the application of systems theory to product development. There is some overlap and synergy between the disciplines of systems analysis, systems architecture, and systems engineering.

3.1 Database Design (ER Model)

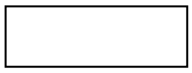
Database Designing is a part of the development process. In the linear development cycle, it is used during the system requirements phase to construct the data components of the analysis model. This model represents the major data objects and the relationship between them. It should not be confused with data analysis, which takes place in the system design phase. As in a DFD, a model of data consists of a number of symbols joined up according to certain conventions. System designers describe these conceptual modeling using symbols from a modeling method known as entity relationship analysis.

Entity Relationship Diagram

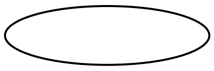
Entity relationship analysis uses three major abstractions to describe data. These are

- ❖ Entities, which are distinct things in the enterprise.
- ❖ Relationships, which are meaningful interactions between the objects.
- ❖ Attributes, which are the properties of the entities and relationships.
- ❖ The relative simplicity and pictorial clarity of this diagramming technique may well account in large part for the widespread use of ER model. Such a diagram consists of the following major components:

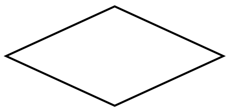
ER Diagram Components



Rectangles, which represent the entity set.



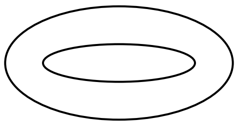
Ellipse, which represent attributes.



Diamonds, which represent relationship sets.

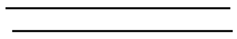
Lines, which link attributes to entity sets and

entity sets to relationships.



Double Ellipse, which represents multivalued

Double lines, which indicates participation of an entity in a relationship set.



Entity

- ❖ An entity is an object that exists and is distinguishable from other objects.
- ❖ An entity may be concrete or abstract.
- ❖ An entity is a set of entities of the same type.
- ❖ Entity sets need not be disjoint.
- ❖ An entity is represented by a set of attributes.

Mapping Constraints

An E-R diagram may define certain constraints which the contents of a database must conform.

Mapping Cardinalities

It expresses the number of entities to which another entity can be associated via a relationship. For binary relationship sets between entity sets A and B, the mapping cardinality must be one of the following:

One-to-One – An entity in A is associated with at most one entity in B, and an entity in B is associated with at most one entity in A.

One-to-many – An entity in A is associated with any number in B. An entity in B is associated with any number in A.

Many-to-many – Entities in A and B are associated with any number from each other.

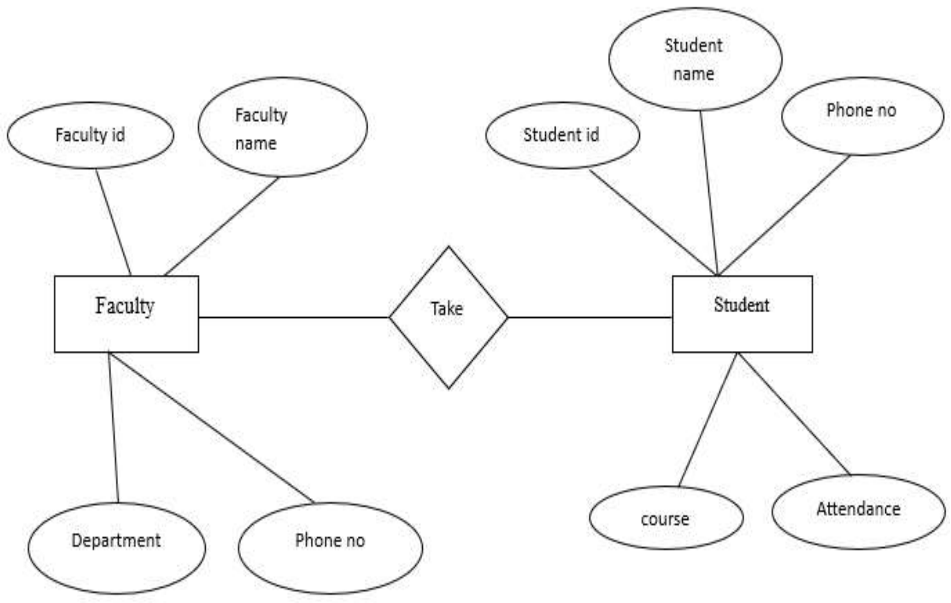


Fig 3.1.1: ER Diagram for Overall System

3.2 Data Dictionary

The logical characteristics of current systems data stores, including name, description, aliases, contents, and organization, identifies processes where the data are used and where immediate access to information required, Serves as the basis for identifying database requirements during system design.

Uses of Data Dictionary

- ❖ To manage the details in large systems.
- ❖ To communicate a common meaning for all system elements.
- ❖ To Document the features of the system.

Table Name: Faculty

Column Name	**Data Type (Size)**	**Constraints**
Faculty Id	Number(10)	PRIMARY
Faculty Name	Varchar2(30)	NOTNULL
Phone no	Number(10)	NOTNULL
Department	Varchar2(30)	NOTNULL

Table 3.2.1: Faculty

Description: This table shows faculty details.

Table Name: Student

Column Name	**Data Type (Size)**	**Constraints**
Student id	Number(10)	PRIMARY
Student name	Varchar2(30)	NOTNULL
Phone no	Number(10)	NOTNULL
Course	Varchar2(30)	NOTNULL
Attendance	Number(10)	NOTNULL

Table 3.2.2: Student

Description: This table shows student details.

3.3 UML Design

To understand the UML, you need to form a conceptual model of the language, and this requires learning three major elements: the UML's basic building blocks, the rules that dictate how these building blocks may be put together, and some common mechanisms that apply throughout the UML. Once you have grasped these ideas, you will be able to read UML models and create some basic ones. As you gain more experience in applying the UML, you can build on this conceptual model by using more advanced features of the language.

Building Blocks of the UML

The vocabulary of the UML encompasses three kinds of building blocks.

1. Things
2. Relationships
3. Diagrams

Things in the UML

Things are the abstractions that are first-class citizens in a model; relationships tie these things together, and diagrams group interesting collections of things. There are four kinds of things in the UML.

1. Structural things
2. Behavioural things
3. Grouping things
4. Annotational things

Structural things

Structural things are the nouns of UML models. These are the mostly static parts of a model, representing elements that are either conceptual or physical. In all, there are seven kinds of structural things.

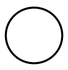
Classes

First, a class is a description of a set of objects that share the same attributes, operations, relationships, and semantics. A class implements one or more interfaces. Graphically, a class is rendered as a rectangle, usually including its name, attributes, and operations.

Windows
Origin
Open()

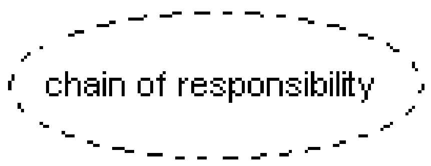
Interface

Second, an interface is a collection of operations that specify the service of a class or component. An interface therefore describes the externally visible behaviour of that element. An interface might represent the complete behaviour of a class or component or only a part of that behaviour. An interface defines a set of operation specifications. Graphically, an interface is rendered as a circle with its name. An interface is rarely rendered alone. Rather, it is typically attached to the class or component that realises the interface.



Collaborations

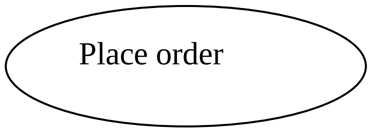
Third, collaboration defines an interaction and is a society of roles and other elements that work together to provide some cooperative behaviour that's bigger than the sum of all the elements. Therefore, collaborations have structural as well as behavioural dimensions. A given class might participate in several collaborations; these collaborations therefore represent the implementation of patterns that make up a system. Graphically, collaboration is rendered as an ellipse with dashed lines, usually including only its name.



Use Cases

Fourth, a use case is a description of a set of sequences of actions that a system performs that yield an observable result of value to a particular actor.

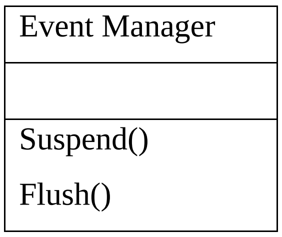
A use case is used to structure the behavioural aspects of a model. A use case is realised through collaboration. Graphically, the use case is rendered as an ellipse with solid lines, usually including only its name.



The remaining three things, active classes, components, and nodes, are all class-like, meaning they also describe a set of objects that share the same attributes, operations, relationships, and semantics. However, these three are different enough and are necessary for modelling certain aspects of an object-oriented system, so they warrant special treatment.

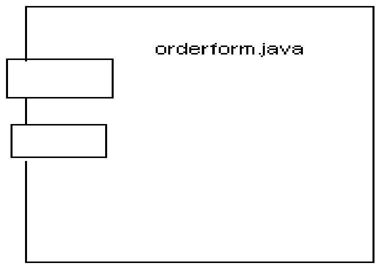
Active Classes

Fifth, an active class is a class whose objects own one or more processes or threads and therefore can initiate control activity. An active class is just like a class except that its objects represent elements whose behaviour is concurrent with other elements. Graphically, an active class is rendered just like a class but with heavy lines, usually including its name, attributes, and operations. The remaining two elements, components and nodes, are also different. They represent physical things, whereas the previous five things represent conceptual things.



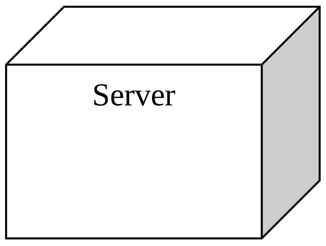
Components

Sixth, a component is a physical and replaceable part of a system that conforms to and provides the realisation of a set of interfaces. In a system, you'll encounter different kinds of deployment components, such as COM+ components or Java beans, as well as components that are artefacts of the development process, such as source code files. A component typically represents the physical packaging of otherwise logical elements, such as classes, interfaces, and collaborations. Graphically, a component is rendered as a rectangle with tabs, usually including only its name.



Nodes

Seventh, a node is a physical element that exists at runtime and represents a computational resource, generally having at least some memory and, often, processing capability. A set of components may reside on a node and may also migrate from node to node. Graphically, a node is rendered as a cube, usually including only its name.

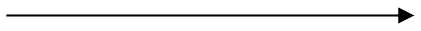


These seven elements classes, interfaces, collaborations, use cases, active classes, components, and nodes are the basic structural things that you may include in a UML model. There are also variations on these seven, such as actors, signals, and utilities (kinds of classes), processes and threads (kinds of active classes), and applications, documents, files, libraries, pages, and tables (kinds of components).

Behavioural Things

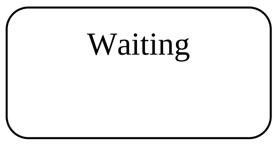
Behavioural things are the dynamic parts of UML models. These are the verbs of a model representing behaviour over time and space. All in all, there are two primary kinds of behavioural things.

First, an interaction is a behaviour that comprises a set of messages exchanged among a set of objects within a particular context to accomplish a specific purpose. The behaviour of a society of objects or an individual operation may be specified with an interaction. An interaction involves a number of other elements, including messages, action sequences (the behaviour invoked by a message), and links (the connection between objects). Graphically, the message is rendered as a directed line, almost always including the name of its operation.



Display Messages

Second, a state machine is a behaviour that specifies the sequence of states an object or an interaction goes through during its lifetime in response to events, together with its responses to those events. The behaviour of an individual class or a collaboration of classes may be specified with a state machine. A state machine involves a number of other elements, including states, transitions (the flow from state to state), events (things that trigger a transition), and activities (the response to a transition). Graphically, a state is rendered as a rounded rectangle, usually including its name and its substates.



States

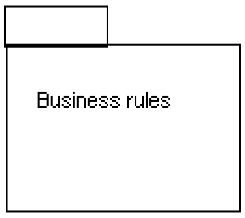
These two elements—interactions and state machines—are the basic behaviours that you may include in a UML model. Semantically, these elements are usually connected to various structural elements, primarily classes, collaborations, and objects.

Grouping Things

Grouping things is one of the original parts of UML models. These are the boxes into which a model can be decomposed. All in all, there is one primary kind of grouping, namely, packages.

Packages

A package is a general-purpose mechanism for organising elements into groups. Structural things, behavioural things, and even other groupings may be placed in a package. Unlike components (which exist at run time), a package is purely conceptual (meaning that it exists only at development time). Graphically, a package is rendered as a tabbed folder, usually including only its name and, sometimes, its contents.



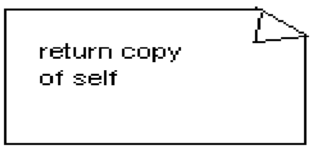
Packages are the basic grouping of things with which you may organise a UML model; there are also variations, such as frameworks, models, and subsystems (kinds of packages).

Annotational Things

Annotational things are the explanatory parts of the UML model. These are the comments you may apply to describe, illuminate, and remark about any element in a model. There is one primary kind of annotational thing called a note. A note is simply a symbol for rendering constraints and comments attached to an element or a collection of elements.

Notes

This element is the one basic annotational thing you may include in a UML model. You'll typically use notes to adorn your diagrams with constraints or comments that are best expressed in informal or formal text. There are also variations on this element, such as requirements (which specify some desired behaviour from the perspective of someone outside the model).

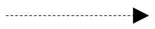


Relationships in the UML

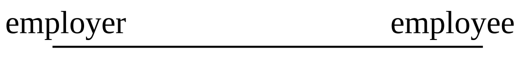
There are four kinds of relationships in the UML.

1. Dependency
2. Association
3. Generalisation
4. Realisation

These relationships are the basic relational building blocks of the UML. You use them to write well-formed models. First, a dependency is a semantic relationship between two things in which a change to one thing (the independent thing) may affect the semantics of the other thing (the dependent thing). Graphically, dependency is rendered as a dashed line, possibly directed, and occasionally including a label.

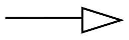


Second, an association is a structural relationship that describes a set of links, a link being a connection among objects. Aggregation is a special kind of association, representing a structural relationship between a whole and its parts. Graphically, an association is rendered as a solid line, possibly directed, occasionally including a label, and often containing other adornments, such as multiplicity and role names.

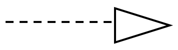


Associations

Third, a generalisation is a specialisation-generalisation relationship in which objects of the specialised element (the child) are substitutable for objects of the generalised element (the parent). In this way, the child shares the structure and behaviour of the parent. Graphically, a generalisation relationship is rendered as a solid line with a hollow arrowhead pointing to the parent.



Fourth, a realisation is a semantic relationship between classifiers wherein one classifier specifies a contract that another classifier guarantees to carry out. You'll encounter realisation relationships in two places: between interfaces and the classes or components that realise them, and between use cases and the collaborations that realise them. Graphically, a realisation relationship is rendered as a cross between a generalisation and a dependency relationship.



These four elements are the basic relational things you may include in a UML model. There are also variations on these four, such as refinement, trace, include, and extended (for dependencies). A UML system is represented using five different views that describe the system from distinctly different perspectives. Each view is defined by a set of diagrams, which are as follows:

User Model View

This view represents the system from the user's perspective. The analysis representation describes a usage scenario from the end-user's perspective.

S**tructural Model View**

In this model, the data and functionality come from inside the system. This model view models the static structures.

Behavioural Model View

It represents the dynamic of behavioural elements as parts of the system, depicting the interactions of collection between various structural elements described in the user model and structural model view.

Implementation Model View

In this structural and behavioural view, parts of the system are represented as they are to be built.

Environmental Model View

In this, the structural and behavioural aspects of the environment in which the system is to be implemented are represented.

Diagrams in the UML

A diagram is the graphical presentation of a set of elements, most often rendered as a connected graph of vertices (things) and arcs (relationships). You draw diagrams to visualise a system from different perspectives, so a diagram represents an elided view of the elements that make up a system. The same element may appear in all diagrams, only a few diagrams, or in no diagrams at all. In theory, a diagram may contain any combination of things and relationships. In practise, however, a small number of common combinations arise that are consistent with the five most useful views that comprise the architecture of a software-intensive system. For this reason, the UML includes nine such diagrams.

1. Use case Diagram
2. Class Diagram
3. Sequence Diagram
4. Collaboration Diagram
5. Component Diagram
6. Deployment Diagram
7. State chart Diagram
8. Activity Diagram

Use Case Diagram

Use Case Diagrams describe what a system does from the standpoint of an external observer. Use-case diagrams are closely connected to scenarios. A scenario is an example of what happens when someone interacts with a system. A use case is a summary of scenarios for a single task or goal. An actor is someone or something who initiates the events involved in that task.

Sequence Diagram

A sequence diagram is an interaction diagram that details how operations are carried out, i.e., what messages are sent and when. A sequence diagram represents the timely ordering of messages. The objectives involved in the operation are listed from left to right according to when they take part in the message sequence.

Class Diagram

A class diagram gives an overview of a system by showing its classes and the relationships among them. Class diagrams are static—they display what interacts but not what happens when they do interact.

State Chart Diagram

Objects have behaviours and states. The state of an object depends on its current activity or condition. A state chart diagram shows the possible states of the object and the transitions that cause a change in state. States are rounded rectangles. Transitions are arrows from one state to another. Events are conditions that trigger transitions and are written beside the arrows.

Activity Diagram

An activity diagram is essentially a fancy flowchart. Activity diagrams and state chart diagrams are related. While a state chart diagram focuses attention on an object undergoing a process (or on a process as an object), an activity diagram focuses on the flow of activities involved in a single process.

Use Case Diagram

Use case diagrams to graphically depict system behaviour. These diagrams present a high-level view of how the system is used as viewed from an outsider’s (actor’s) perspective. A use-case diagram may depict all or some of the use cases of a system.

A use case diagram can contain:

- Actors
- Use cases

Interaction or relationship between actors and use cases in the system, including associations, dependencies, and generalisations. A use-case diagram can be used during analysis to capture the system requirements and understand how the system should work. During the design phase, you can use use-case diagrams to specify the behaviour of the systems implemented.

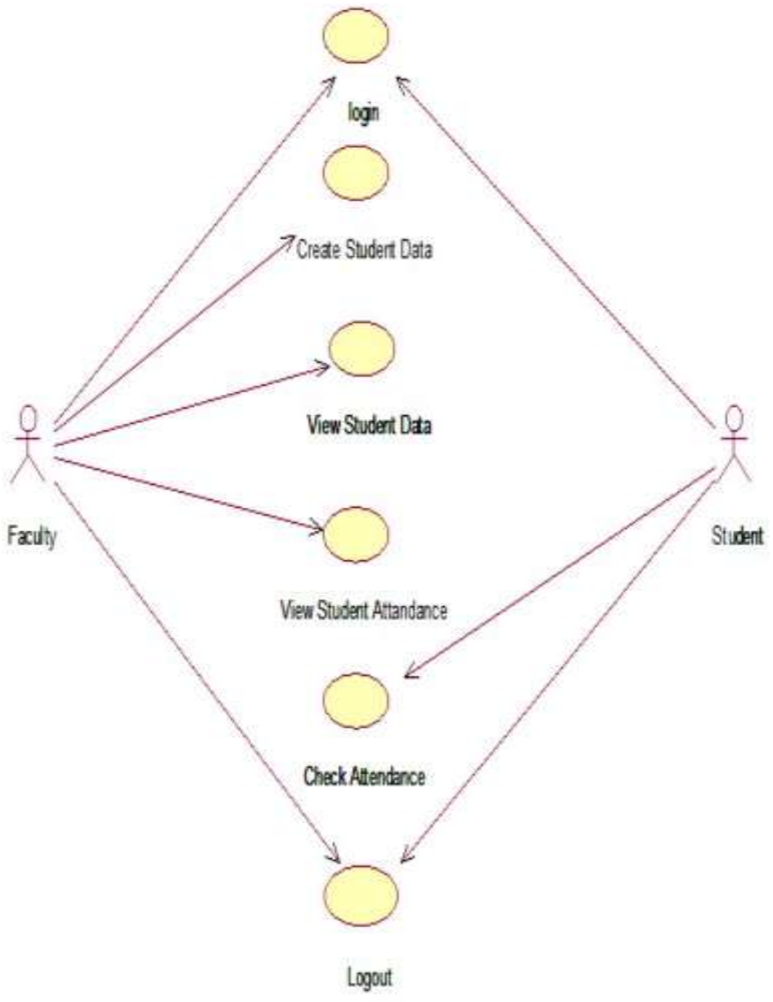


Fig 3.3.1: Use Case Diagram for Overall System

Class Diagram

The UML class diagram also referred to as object modelling, is the main static analysis diagram. Object modelling is the process by which the logical objects in the problem space are represented by the actual objects in the programme. These diagrams show a set of classes, interfaces, and collaborations and their relationships. These diagrams address the static design view of a system. This view primarily supports the functional requirements of a system – the services the system should provide to its end users. A class diagram is a collection of static modelling elements, such as classes and their relationships, connected as a graph to each other and to their contents.

Class diagrams commonly contain the following things:

- Classes
- Interfaces
- Collaborations
- Dependency, Generalisation and association relationships

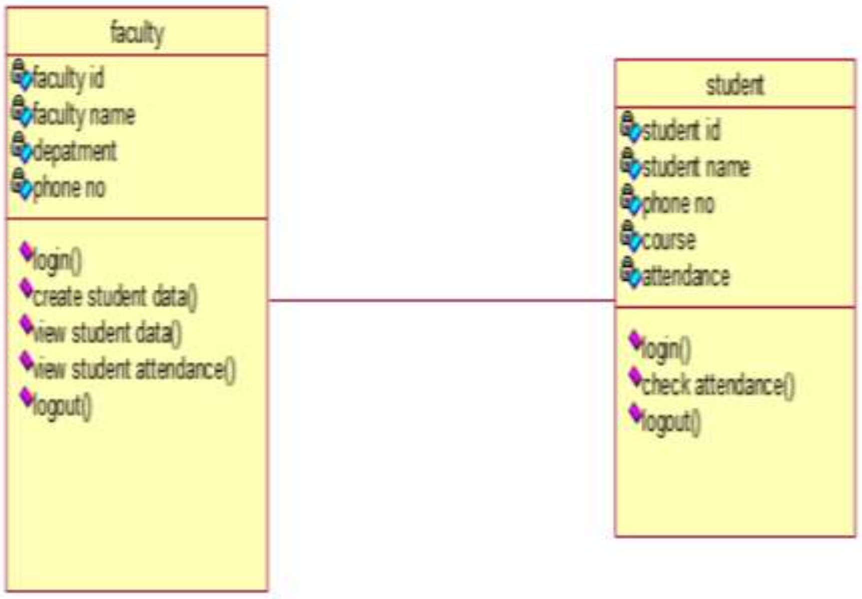


Fig 3.3.2: Class Diagram for Overall System

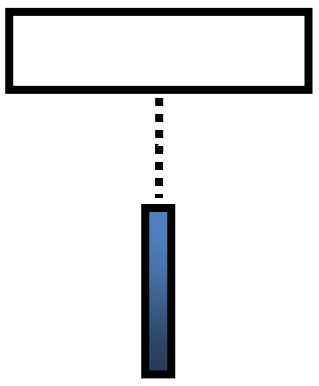
Sequence Diagram

It is an interaction diagram that emphasises the time ordering of messages. A sequence diagram shows objects participating in the interaction by their lifetime and the messages that they exchange/arranged in the time sequence.

Description

It contains

- ❖ **Object:** It is represented as horizontal rectangle.
- ❖ **Object Lifeline:** It represents the existence of an object over a period.
- ❖ **Focus of control:** It is a tall, thin rectangle that shows the period during which an object is performing an action.



- ❖ **Messages:** It is communication between objects, shown as horizontal solid arrow from one object to another object.

  In the below diagram

It is a type interaction diagram that shows the interaction between a set of objects. The way they are linked to each other and the message.

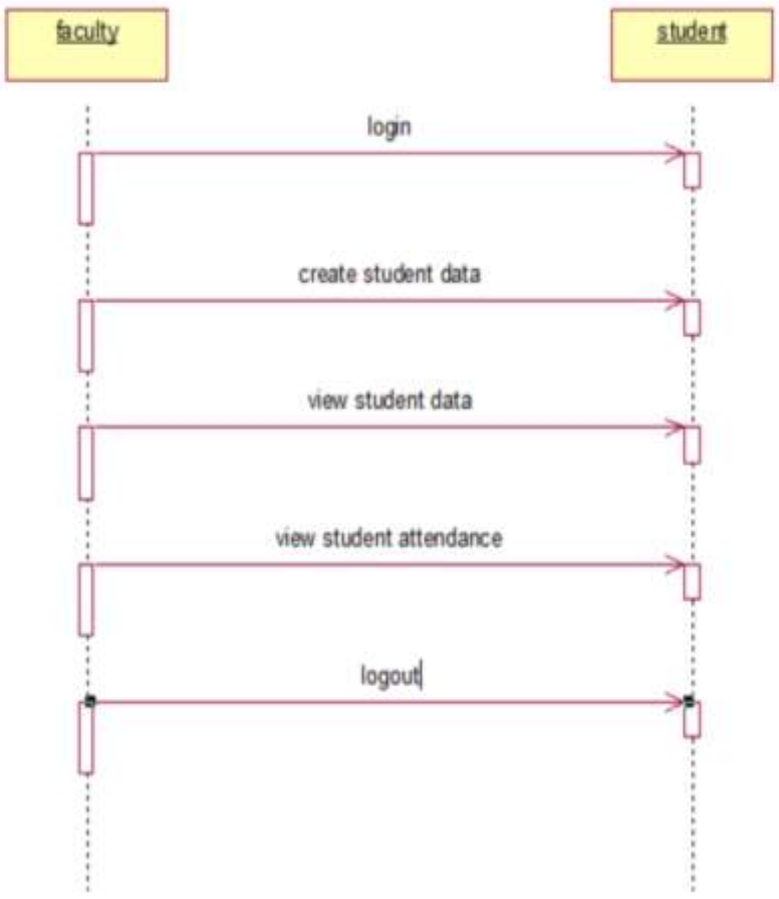


Fig 3.3.3: Sequence Diagram for Faculty

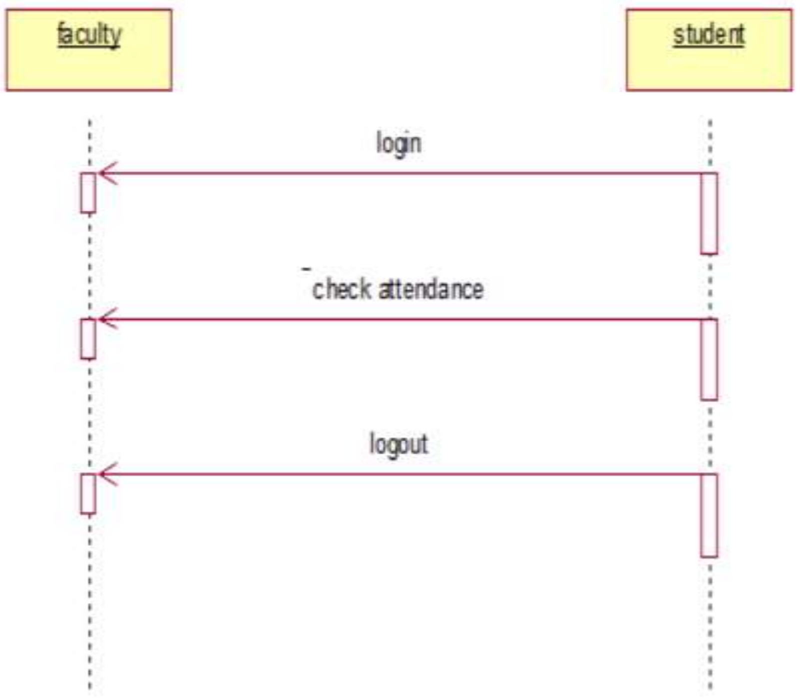


Fig 3.3.4: Sequence Diagram for Student

Activity Diagram

An Activity diagrams illustrates the dynamic nature of a system by modelling the flow of control from activity to activity. An activity represents an operation on some class in the system that results in a change in the state of the system. Typically, activity diagrams are used to model workflow or business processes and internal operation. Because an activity diagram is a special kind of state chart diagram, it uses some of the same modelling conventions.

Basic Activity Diagram Symbols and Notations

Action states

Action states represent the non-interruptible actions of objects. You can draw an action state in Smart Draw using a rectangle with rounded corners.

Action Flow

Action flow arrows illustrate the relationships among action states.

Object Flow

Object flow refers to the creation and modification of objects by activities. An object flow arrow from an action to an object means that the action creates or influences the object.

Initial State

A filled circle followed by an arrow represents the initial action state.

Final State

An arrow pointing to a filled circle nested inside another circle represents the final action state.

Branching

A diamond represents a decision with alternate paths. The outgoing alternates should be labelled with a condition or guard expression. You can also label one of the paths "else'.

Synchronisation

A synchronisation bar helps illustrate parallel transitions. Synchronisation is also called for king and joining.

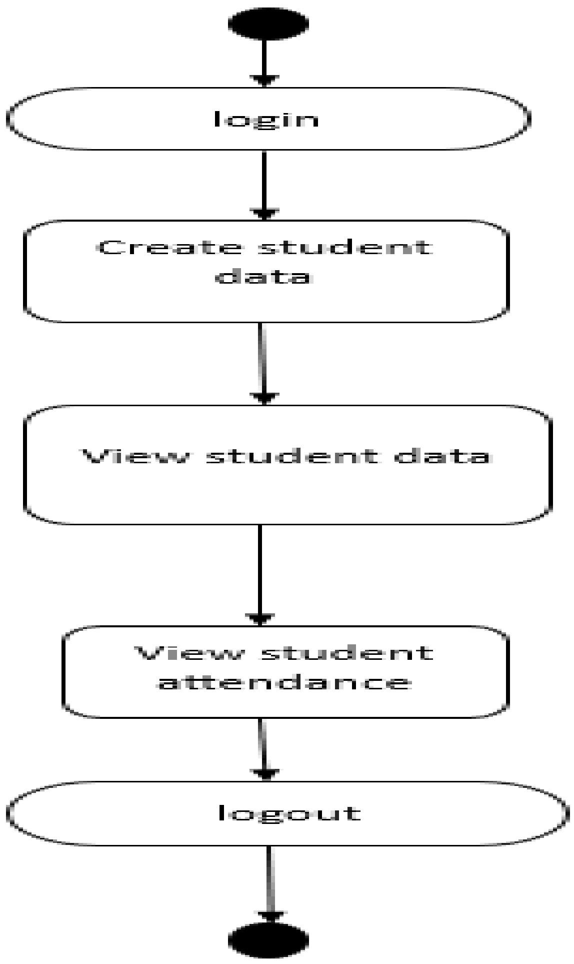


Fig 3.3.5: Activity Diagram for Faculty

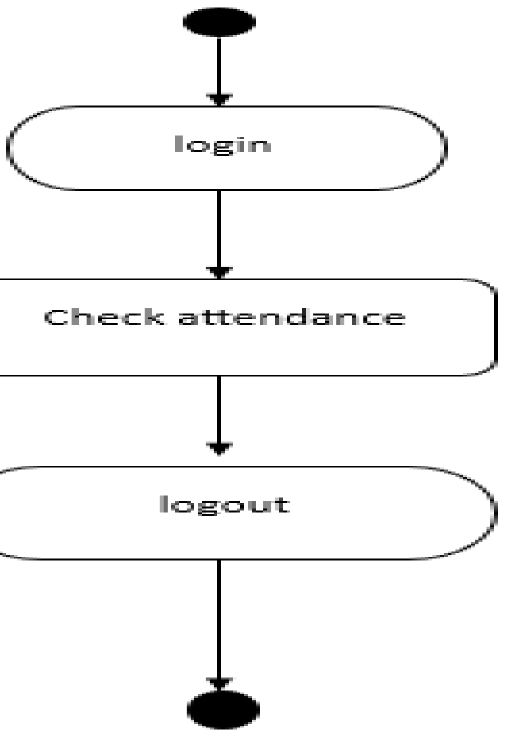


Fig 3.3.6: Activity Diagram for Student

Deployment Diagram

Deployment diagrams describe the configuration of run-time processing resource elements and the mapping of software implementation components on to them. These Diagrams contain components and nodes, which represent processing or computational resources, including computers, printers, etc.

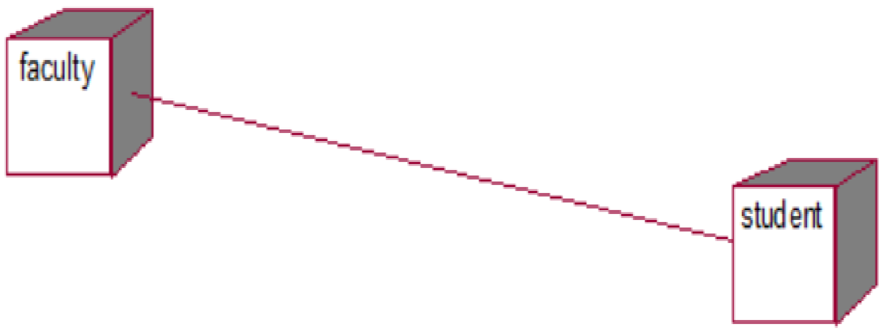


Fig 3.3.7: Deployment Diagram for Overall System