Dr. B. R. Ambedkar National Institute of Technology, Jalandhar

Practical File

Software Engineering Laboratory

CSX - 323

Submitted to :

Dr. Kuldeep Kumar

Assistant Professor

Department of CSE

Team Members :

Abhinendra Gaurav (17103002)

Aditya Garg (17103004)

Ankit Goyal (17103011)

INDEX

4. Papyrus

Eclipse Papyrus is an industrial-grade open source Model-Based Engineering tool. Eclipse Papyrus has notably been used successfully in industrial projects and is the base platform for several industrial modelling tools.
The primary mechanism for specialization of *Papyrus*, especially as it pertains to the implementation of a derivative language of UML, is the extension points provided by the *Papyrus UML* platform. These are always preferred where possible.

Architecture Model

This model defines the identity of the *UML Light* language and, critically, the diagram types (*representation kinds*) that visualize models in that language. So, many of the other extensions flow from these, either directly by reference from the architecture model or indirectly by their activation in the context of that architecture.
For each representation kind in the language, the architecture model provides

- a name, unique identifier, icon, and brief description
- an implementation ID, which in the case of *UML Light* is always the implementation ID of the *Papyrus UML* diagram editor on which the specialized diagram is based
- the command class that creates the diagram. Following the implementation ID, this is the *Papyrus UML* creation command class for the corresponding base diagram

UML 2.5.0

Eclipse Papyrus is graphical editing tool for UML 2 as defined by OMG. Eclipse Papyrus targets to implement 100% of the OMG specification!

Eclipse Papyrus provides editors for all the UML diagrams:

- Class Diagram
- Object Diagram
- Package Diagram
- Composite Structure Diagram
- Component Diagram
- Deployment Diagram
- Profile Diagram
- Use case Diagram
- Activity Diagram
- State machine Diagram
- Communication Diagram

- Sequence Diagram
- Timing Diagram
- Interaction overview Diagram

Eclipse Papyrus provides also a complete support to SysML in order to enable model-based system engineering. Specific tabular and graphical editors required for SysML are also provided:

- Block Definition Diagram
- Internal Block Diagram
- Requirement Diagram
- Parametric Diagram
- Requirement table
- Allocation table

Eclipse Papyrus can execute models using a rich and extensible animation and simulation framework.

Also, as graphical modelling is not always the best way for specifying the behaviour of executable models, Eclipse Papyrus provides textual notation edition with syntax highlight, completion and content assist. It is of course a customizable feature of Eclipse Papyrus.

Fully customizable environment

All the modelling features of Eclipse Papyrus are designed to be customizable and to maximize reuse. Therefore, should you want to adapt the standard configuration for a specific domain, notation, modelling practice or use the powerful customization mechanisms of Eclipse Papyrus to adapt the modelling environment to suit your needs. Many configurations in Eclipse Papyrus being model-based, the customization can be done live.

- Define your own graphical, textual or tabular notation.
- Filter existing palettes or define your own ones with a model-based configuration.
- Define dedicated properties views to present just the characteristics that are important to you.
- Read your model with dedicated model explorer structuring and rendering.

Reuse standard languages or define your own modelling language thanks to the UML profile editor.

5. Eclipse

In the context of computing, Eclipse is an integrated development environment (IDE) for developing applications using the Java programming language and other programming languages such as C/C++, Python, PERL, Ruby etc.

The Eclipse platform which provides the foundation for the Eclipse IDE is composed of plug-ins and is designed to be extensible using additional plug-ins. Developed using Java, the Eclipse platform can be used to develop rich client applications, integrated development environments and other tools. Eclipse can be used as an IDE for any programming language for which a plug-in is available.

The Java Development Tools (JDT) project provides a plug-in that allows Eclipse to be used as a Java IDE, PyDev is a plugin that allows Eclipse to be used as a Python IDE, C/C++ Development Tools (CDT) is a plug-in that allows Eclipse to be used for developing application using C/C++, the Eclipse Scala plug-in allows Eclipse to be used an IDE to develop Scala applications and PHP. Eclipse is a plug-in to eclipse that provides complete development tool for PHP.

Eclipse got its start in 2001 when IBM donated three million lines of code from its Java tools to develop an open source integrated development environment (IDE). The IDE was initially overseen by a consortium of software vendors seeking to create and foster a new community that would complement Apache's open source community. Rumour has it that the platform's name was derived from a secondary goal, which was to eclipse Microsoft's popular IDE, Visual Studio.

Today, Eclipse is managed by the Eclipse Foundation, a not-for-profit corporation whose strategic members include CA Technologies, IBM, Oracle and SAP. The foundation, which was created in 2004, supports Eclipse projects with a well-defined development process that values quality, application programming interface (API) stability and consistent release schedules. The foundation provides infrastructure and intellectual property (IP) management services to the Eclipse community and helps community members market and promote commercial software products that are based on Eclipse.

6. Use Case Diagram

A use case diagram is the primary form of system/software requirements for a new software program underdeveloped. Use cases specify the expected behaviour, and not the exact method of making it happen (how). Use cases once specified can be denoted both textual and visual representation (i.e. use case diagram). A key concept of use case modelling is that it helps us design a system from the end user's perspective. It is an effective technique for communicating system behaviour in the user's terms by specifying all externally visible system behaviour.
A use case diagram is usually simple. It does not show the detail of the use cases:

- It only summarizes **some of the relationships** between use cases, actors, and systems.
- It does **not show the order** in which steps are performed to achieve the goals of each use case.

Purpose of Use Case Diagrams

The purpose of use case diagram is to capture the dynamic aspect of a system. However, this definition is too generic to describe the purpose, as other four diagrams (activity, sequence, collaboration, and State chart also have the same purpose. We will look into some specific purpose, which will distinguish it from other four diagrams.

Use case diagrams are used to gather the requirements of a system including internal and external influences. These requirements are mostly design requirements. Hence, when a system is analysed to gather its functionalities, use cases are prepared and actors are identified.

When the initial task is complete, use case diagrams are modelled to present the outside view.

In brief, the purposes of use case diagrams can be said to be as follows –

- Specify the context of a system
- Capture the requirements of a system
- Validate a systems architecture
- Drive implementation and generate test cases
- Developed by analysts together with domain experts

NOTATIONS:-

- **Actor**
  - Someone interacts with use case (system function).
  - Named by noun.

- Actor plays a role in the business
- Similar to the concept of user, but a user can play different roles

- **Use Case**
  - System function (process - automated or manual)
  - Named by verb + Noun (or Noun Phrase).

- **Communication Link**
  - The participation of an actor in a use case is shown by connecting an actor to a use case by a solid link.
  - Actors may be connected to use cases by associations, indicating that the actor and the use case communicate with one another using messages.
- **Boundary of system**
  - The system boundary is potentially the entire system as defined in the requirements document.
  - For large and complex systems, each module may be the system boundary.

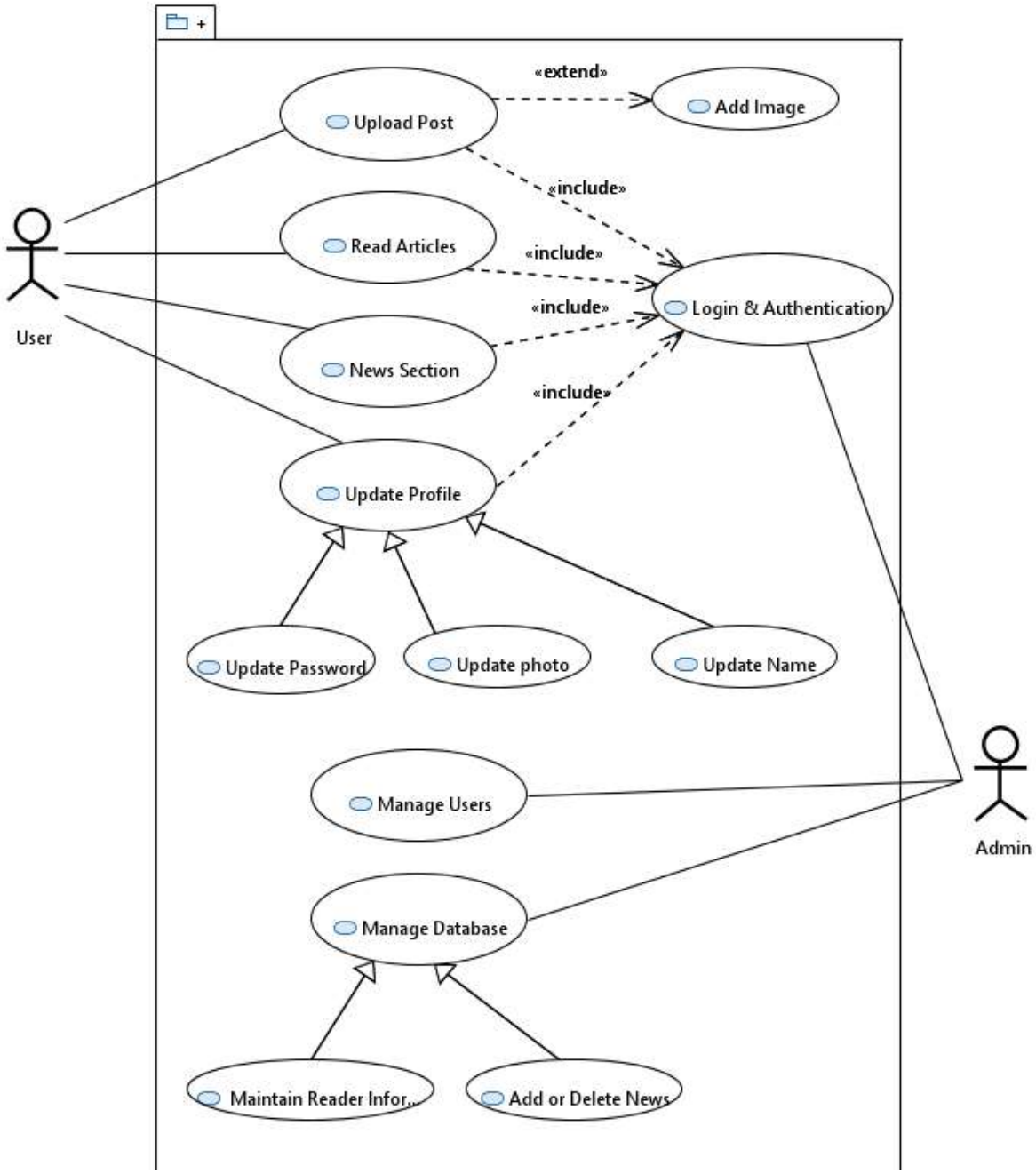

Fig. 6.1 : Use Case Diagram

In the shown use case diagram, the user can do multiple tasks like Upload Post, Read Article, Update Profile. User is able to do so only after Login and Authentication. The user can Update the Profile photo, UserName and the Password. He can upload the posts which contains the title and description of the post. The image field is optional in the post. In the News Section user can see the posts and the articles uploaded by other users. The another actor for the system is a admin. The admin can authenticate the users, manage database and add or delete the users.

7. Class Diagram

Class diagram is a static diagram. It represents the static view of an application. Class diagram is not only used for visualizing, describing, and documenting different aspects of a system but also for constructing executable code of the software application.

Class diagram describes the attributes and operations of a class and also the constraints imposed on the system. The class diagrams are widely used in the modelling of object-oriented systems because they are the only UML diagrams, which can be mapped directly with object-oriented languages.

Class diagram shows a collection of classes, interfaces, associations, collaborations, and constraints. It is also known as a structural diagram.

Purpose of Class Diagram

The purpose of class diagram is to model the static view of an application. Class diagrams are the only diagrams which can be directly mapped with object-oriented languages and thus widely used at the time of construction.

UML diagrams like activity diagram, sequence diagram can only give the sequence flow of the application, however class diagram is a bit different. It is the most popular UML diagram in the coder community.

The purpose of the class diagram can be summarized as –

- Shows static structure of classifiers in a system
- Diagram provides a basic notation for other structure diagrams prescribed by UML
- Helpful for developers and other team members too
- Business Analysts can use class diagrams to model systems from a business perspective

A UML class diagram is made up of:

- A set of classes and
- A set of relationships between classes

What is a Class?

A description of a group of objects all with similar roles in the system, which consists of:

- **Structural features** (attributes) define what objects of the class "know"
  - Represent the state of an object of the class
  - Are descriptions of the structural or static features of a class
- **Behavioural features** (operations) define what objects of the class "can do"
  - Define the way in which objects may interact
  - Operations are descriptions of behavioural or dynamic features of a class

Class Notation

A class notation consists of three parts:

- **Class Name**
  - The name of the class appears in the first partition.
- **Class Attributes**
  - Attributes are shown in the second partition.
  - The attribute type is shown after the colon.
  - Attributes map onto member variables (data members) in code.
- **Class Operations** (Methods)
  - Operations are shown in the third partition. They are services the class provides.
  - The return type of a method is shown after the colon at the end of the method signature.
  - The return type of method parameters is shown after the colon following the parameter name.
  - Operations map onto class methods in code

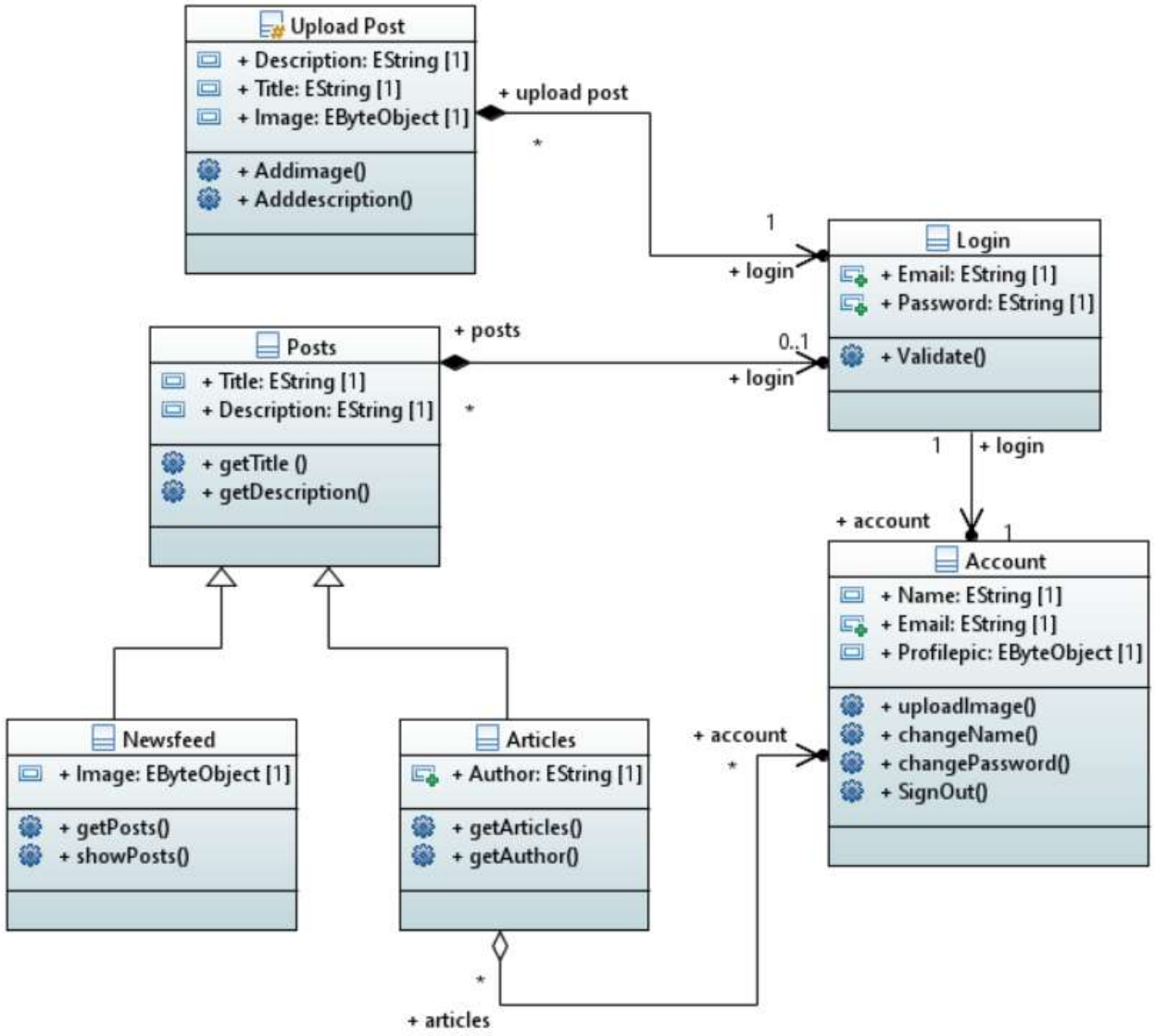

Fig. 7.1 : Class Diagram

In the shown diagram, there is the Login class having variables Email id and Password. It has function for authenticating the user details. In Post class the variables are title and description of the post. There are two functions for getting the title and description of the post. Similarly, in the descendent class NewsFeed there is another function which takes input for the image also. In the Articles class a variable for taking input for Auther is added.

In class UploadPost there are variables to take input from the user related to the post. A single user can upload multiple posts or articles. The user is also able to see the articles of other users.

8. Activity Diagram

Activity Diagrams describe how activities are coordinated to provide a service which can be at different levels of abstraction. Typically, an event needs to be achieved by some operations, particularly where the operation is intended to achieve a number of different things that require coordination, or how the events in a single use case relate to one another, in particular, use cases where activities may overlap and require coordination. It is also suitable for modelling how a collection of use cases coordinates to represent business workflows.

- Identify candidate use cases, through the examination of business workflows
- Identify pre- and post-conditions (the context) for use cases
- Model workflows between/within use cases
- Model complex workflows in operations on objects
- Model in detail complex activities in a high level activity Diagram

The basic purposes of activity diagrams is similar to other four diagrams. It captures the dynamic behavior of the system. Other four diagrams are used to show the message flow from one object to another but activity diagram is used to show message flow from one activity to another.

Activity is a particular operation of the system. Activity diagrams are not only used for visualizing the dynamic nature of a system, but they are also used to construct the executable system by using forward and reverse engineering techniques. The only missing thing in the activity diagram is the message part.

It does not show any message flow from one activity to another. Activity diagram is sometimes considered as the flowchart. Although the diagrams look like a flowchart, they are not. It shows different flows such as parallel, branched, concurrent, and single.

The purpose of an activity diagram can be described as −

- Draw the activity flow of a system.
- Describe the sequence from one activity to another.
- Describe the parallel, branched and concurrent flow of the system.

Activity Diagram Notations:

Activity

Is used to represent a set of actions

Action

A task to be performed

Control Flow

Shows the sequence of execution

Object Flow

Show the flow of an object from one activity (or action) to another activity

Initial Node

Portrays the beginning of a set of actions or activities

Activity Final Node

Stop all control flows and object flows in an activity (or action)

Decision Node

Represent a test condition to ensure that the control flow or object flow only goes down one path

Merge Node

Bring back together different decision paths that were created using a decision-node.

Fork Node

Split behaviour into a set of parallel or concurrent flows of activities.

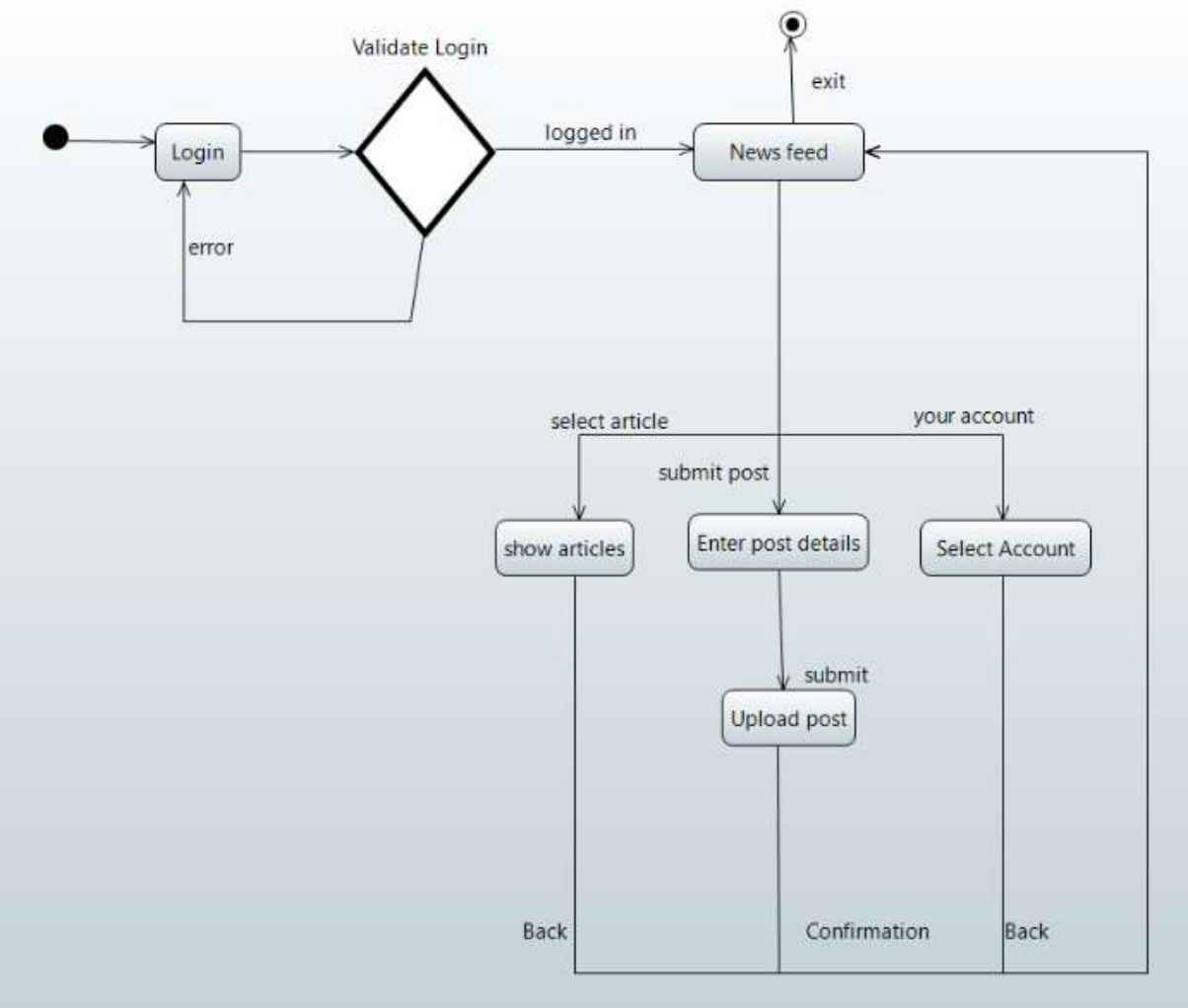

Fig. 8.1 : Activity Diagram

In this diagram, firstly user login within the application, and if the credentials are correct, it will logged in the user. After logging in, the News feed activity will be opened, from where user can select to go to show articles activity, post submission activity, and account activity. These activities can show user the articles, can submit the post by entering details and the, uploading post, confirming and also can access account details. After all this, user can go back to news feed activity, and can exit from there.

9. Sequence Diagram

UML Sequence Diagrams are interaction diagrams that detail how operations are carried out. They capture the interaction between objects in the context of a collaboration. Sequence Diagrams are time focus and they show the order of the interaction visually by using the vertical axis of the diagram to represent time what messages are sent and when.

Sequence Diagrams captures:

- the interaction that takes place in a collaboration that either realizes a use case or an operation.
- high-level interactions between user of the system and the system, between the system and other systems, or between subsystems.

Purpose of Sequence Diagram

- Model high-level interaction between active objects in a system.
- Model the interaction between object instances within a collaboration that realizes a use case.
- Model the interaction between objects within a collaboration that realizes an operation.
- Either model generic or specific instances of a interaction.

Notations

Actor

- a type of role played by an entity that interacts with the subject (e.g., by exchanging signals and data)

Lifeline

- A lifeline represents an individual participant in the Interaction.

Activations

- A thin rectangle on a lifeline) represents the period during which an element is performing an operation.

Call Message

- A message defines a particular communication between Lifelines of an Interaction.

Return Message

- A message defines a particular communication between Lifelines of an Interaction.

Self Message

- A message defines a particular communication between Lifelines of an Interaction.

Recursive Message

- A message defines a particular communication between Lifelines of an Interaction.

Create Message

- A message defines a particular communication between Lifelines of an Interaction.

1. Login Sequence diagram

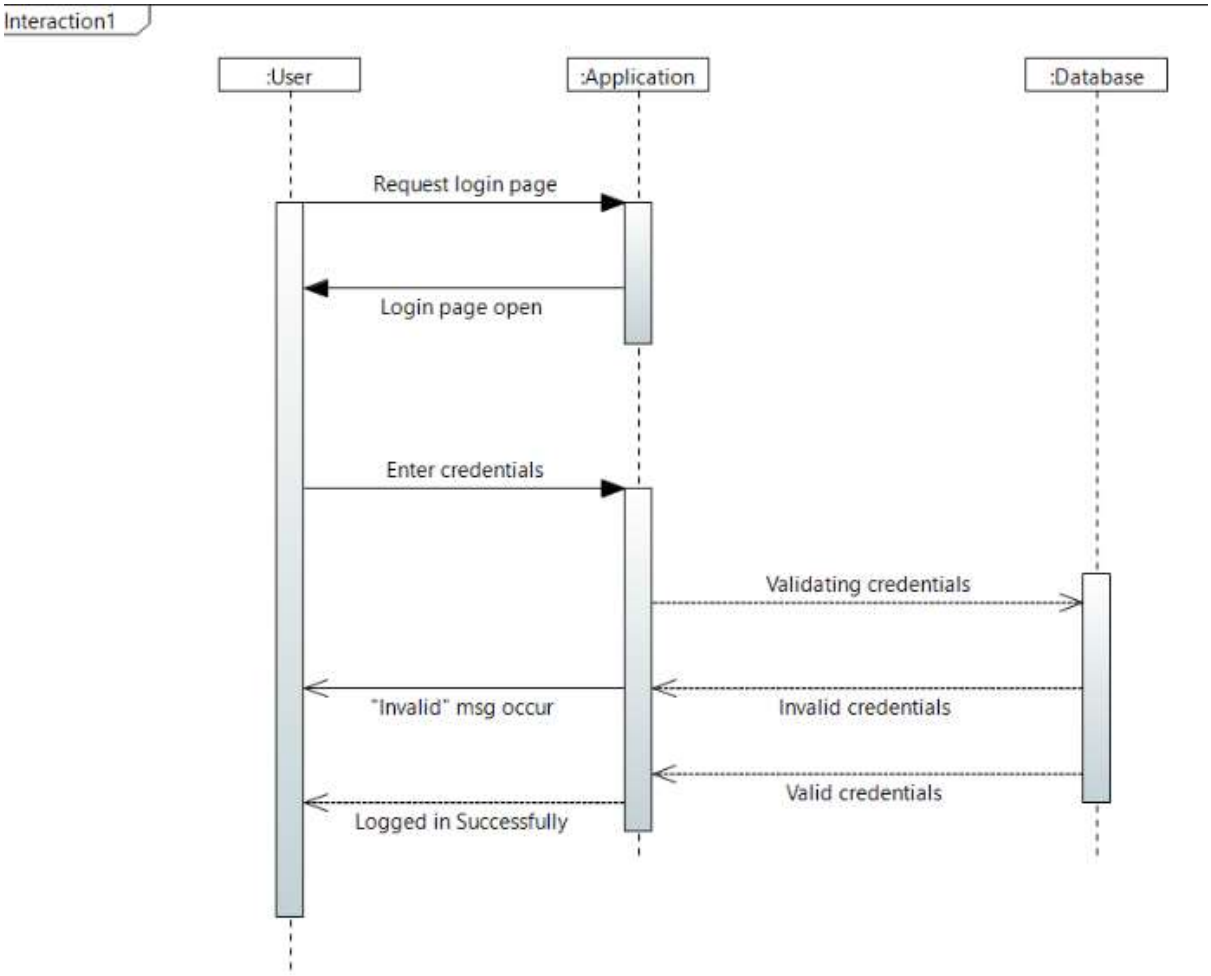

Fig. 9.1 : Sequence diagram Login Feature

In this diagram, User first request the login page to access the authorised application, which it responds by opening login page. User must have filled valid credentials for his account, and hence will use same credentials to validate the account. Once entered, it will validate the credentials from database. If invalid, it will show error message, and return to same login page. If valid, it will logged in successfully to the account.

2. Post Upload Diagram

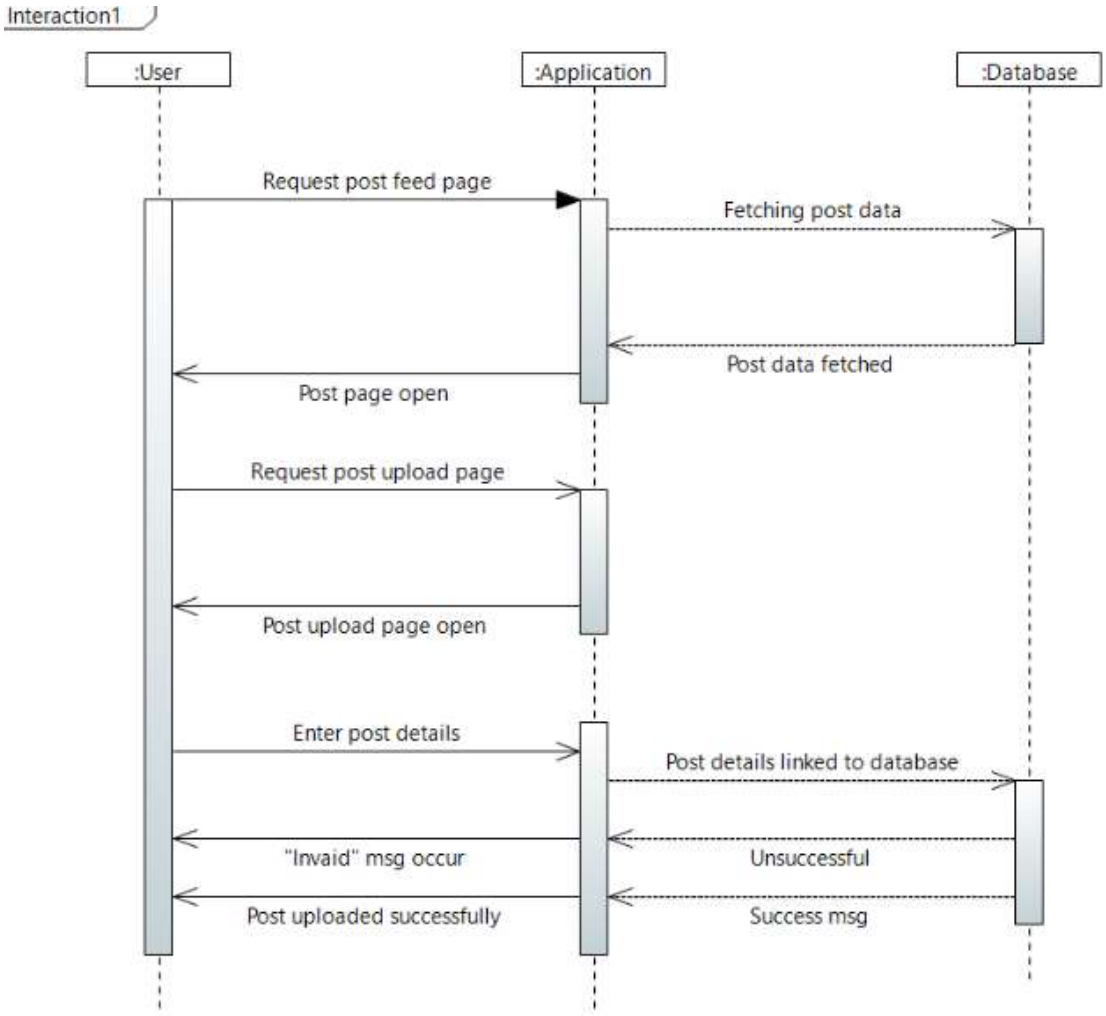

Fig. 9.2 : Sequence Diagram Post Update Feature

In this diagram, User requests post feed page which application responds by opening post page, after getting information and data from Database about other posts. If user wishes to upload any posts, it requests page upload page, which application responds by opening page upload page. User will then enter post details and the post which he wants to upload with valid description, and other requirements. The post will then be linked to database, if uploaded successfully, will be returned to main feed page. Else, user will be given an “invalid” message.

10. State Diagram

State 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 diagrams are useful to model the reactive systems. Reactive systems can be defined as a system that responds to external or internal events.

State 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 diagram is to model lifetime of an object from creation to termination.

State 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 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.

The behavior of an entity is not only a direct consequence of its inputs, but it also depends on its preceding state. The past history of an entity can best be modeled by a finite state machine diagram or traditionally called automata. UML State Machine Diagrams (or sometimes referred to as state diagram, state machine or state chart) show the different states of an entity. State machine diagrams can also show how an entity responds to various events by changing from one state to another. State machine diagram is a UML diagram used to model the dynamic nature of a system.

State machine diagram typically are used to describe state-dependent behaviour for an object. An object responds differently to the same event depending on what state it is in. State machine diagrams are usually applied to objects but can be applied to any element that has behavior to other entities such as: actors, use cases, methods, subsystems systems and etc. and they are typically used in conjunction with interaction diagrams (usually sequence diagrams).

Characteristics of State Machine Notations

There are several characteristics of states in general, regardless of their types:

- A state occupies an interval of time.
- A state is often associated with an abstraction of attribute values of an entity satisfying some condition(s).
- An entity changes its state not only as a direct consequence of the current input, but it is also dependent on some past history of its inputs.

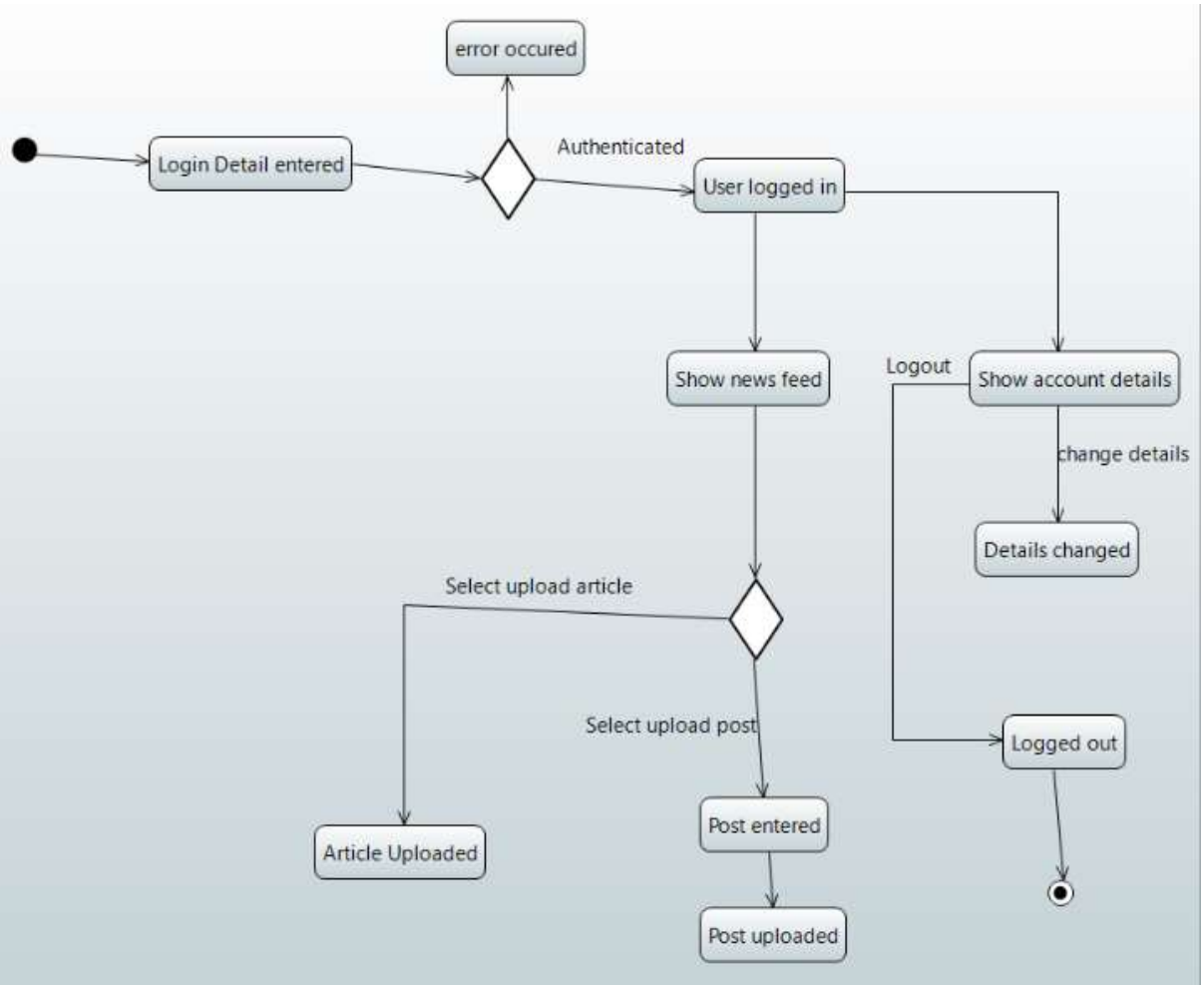

Fig. 10.1 : State Diagram

In the shown state diagram the states of the system is shown in different stages. First the user enters the login details then the admin authenticates the user. After this stage there can be two possibilities. First the user is logged in successfully, second some error occurred. The error can be due to no internet connection or the password entered is wrong. After login the NewsFeed will be displayed. The user can select two options either to upload post or to upload articles. After entering details the post or Article is uploaded to database. In the Account details the user can change the details like Username, Profile photo. The user can select the LogOut option to logout from the app.

11. Collaboration diagram

A collaboration diagram, also known as a communication diagram, is an illustration of the relationships and interactions among software objects in the Unified Modelling Language (UML). These diagrams can be used to portray the dynamic behaviour of a particular use case and define the role of each object. Collaboration diagrams are created by first identifying the structural elements required to carry out the functionality of an interaction. A model is then built using the relationships between those elements. Several vendors offer software for creating and editing collaboration diagrams.

Notations of a collaboration diagram

A collaboration diagram resembles a flowchart that portrays the roles, functionality and behavior of individual objects as well as the overall operation of the system in real time.

The four major components of a collaboration diagram are:

- Objects

  Objects are shown as rectangles with naming labels inside. The naming label follows the convention of object name: class name. If an object has a property or state that specifically influences the collaboration, this should also be noted.

- Actors

  Actors are instances that invoke the interaction in the diagram. Each actor has a name and a role, with one actor initiating the entire use case.

- Links

  Links connect objects with actors and are depicted using a solid line between two elements. Each link is an instance where messages can be sent.

- Messages

  Messages between objects are shown as a labeled arrow placed near a link. These messages are communications between objects that convey information about the activity and can include the sequence number.

The most important objects are placed in the center of the diagram, with all other participating objects branching off. After all objects are placed, links and messages should be added in between.

When to use a collaboration diagram

Collaboration diagrams should be used when the relationships among objects are crucial to display.

A few examples of instances where collaboration diagrams might be helpful include:

- Modeling collaborations, mechanisms or the structural organization within a system design.
- Providing an overview of collaborating objects within an object-oriented system.
- Exhibiting many alternative scenarios for the same use case.
- Demonstrating forward and reverse engineering.
- Capturing the passage of information between objects.
- Visualizing the complex logic behind an operation.

a. Login Feaure

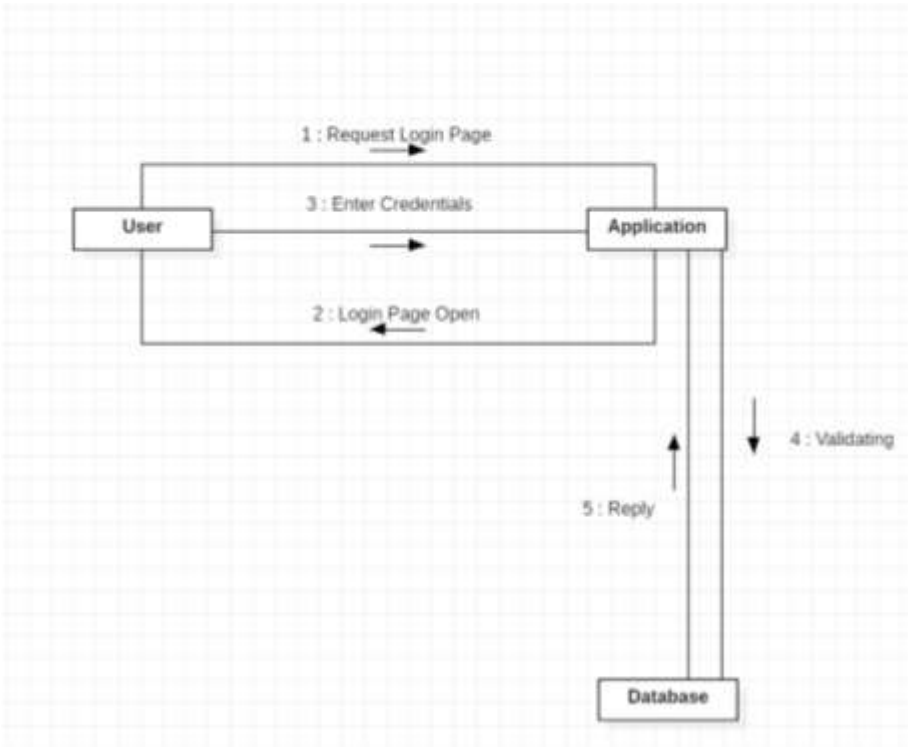

Fig. 11.1 : Collaboration Diagram

This collaboration diagram shows that firstly, the user will request application to show up with login page, and will get replied with opening of login page. The user will enter the valid credentials to login which the application checks from the database, and it will give the appropriate reply. The sequence of messages and processes are shown with the number.

b. Post Upload Feature

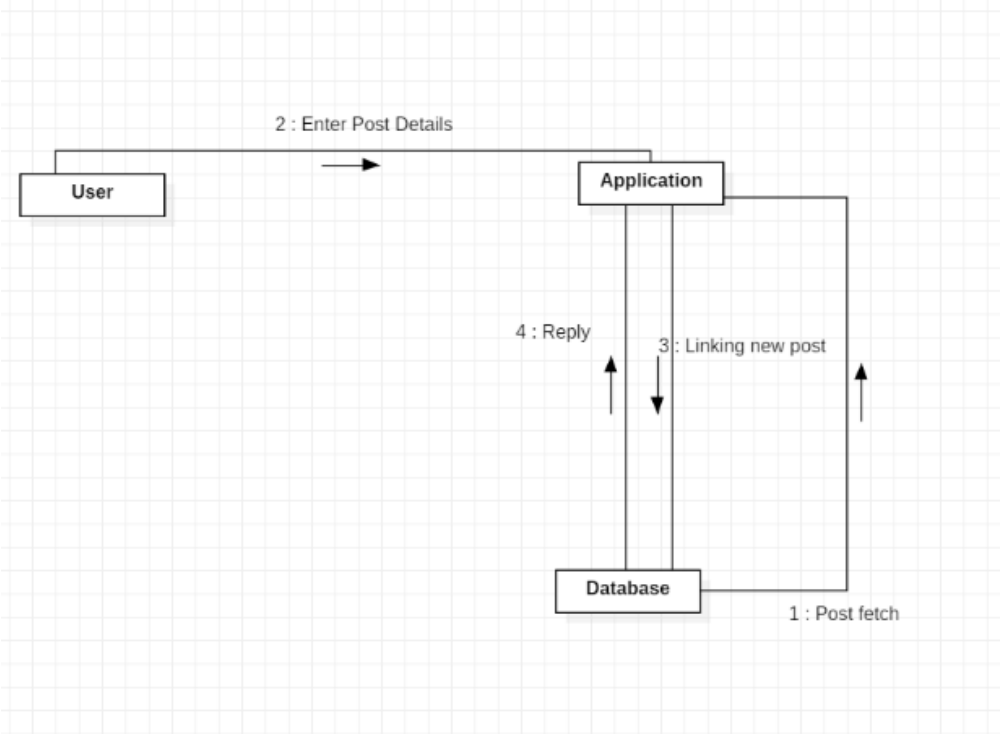

Fig. 11.2 : Collaboration diagram 2

This diagram shows how the post is uploaded, after user request post upload page. User will fill the post details and then, submit it. The post will then be given to link with database and the database will provide appropriate reply to the user.

12. Data Flow Diagram and Structure Chart

Data Flow Diagrams

The DFD (also known as the bubble chart) is a simple graphical formalism that can be used to represent a system in terms of the input data to the system, various processing carried out on those data, and the output data generated by the system. The main reason why the DFD technique is so popular is probably because of the fact that DFD is a very simple formalism— it is simple to understand and use. A DFD model uses a very limited number of primitive symbols to represent the functions performed by a system and the data flow among these functions. Starting with a set of high-level functions that a system performs, a DFD model represents the subfunctions performed by the functions using a hierarchy of diagrams. We had pointed out while discussing the principle of abstraction ,that any hierarchical representation is an effective means to tackle complexity. Human mind is such that it can easily understand any hierarchical model of a system—because in a hierarchical model, starting with a very abstract model of a system, various details of the system are slowly introduced through different levels of the hierarchy. The DFD technique is also based on a very simple set of intuitive concepts and rules. We now elaborate the different concepts associated with building a DFD model of a system.

Structure Chart

A Structure Chart in software engineering and organizational theory is a chart which shows the breakdown of a system to its lowest manageable levels. They are used in structured programming to arrange program modules into a tree. Each module is represented by a box, which contains the module's name. The tree structure visualizes the relationships between modules.

A structure chart is a top-down modular design tool, constructed of squares representing the different modules in the system, and lines that connect them. The lines represent the connection and or ownership between activities and sub-activities as they are used in organization charts.

In structured analysis structure charts, according to Wolber (2009), "are used to specify the high-level design, or architecture, of a computer program. As a design tool, they aid the programmer in dividing and conquering a large software problem, that is, recursively breaking a problem down into parts that are small enough to be understood by a human brain. The process is called top-down design, or functional decomposition. Programmers use a structure chart to build a program in a manner similar to how an architect uses a blueprint to build a house. In the design stage, the chart is drawn and used as a way for the client and the various software designers to communicate. During the actual building of the program (implementation), the chart is continually referred to as "the master-plan".

A structure chart depicts

- the size and complexity of the system, and
- number of readily identifiable functions and modules within each function and
- whether each identifiable function is a manageable entity or should be broken down into smaller components.

A structure chart is also used to diagram associated elements that comprise a run stream or thread. It is often developed as a hierarchical diagram, but other representations are allowable.

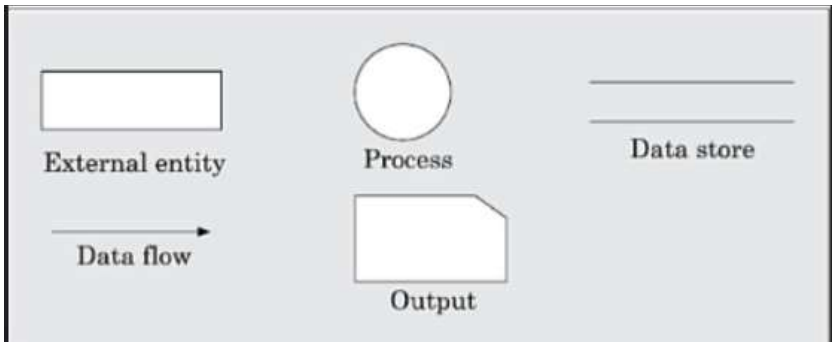

Fig. 12.1 : Entities

HOSPITAL MANAGEMENT SYSTEM

Level 0 :

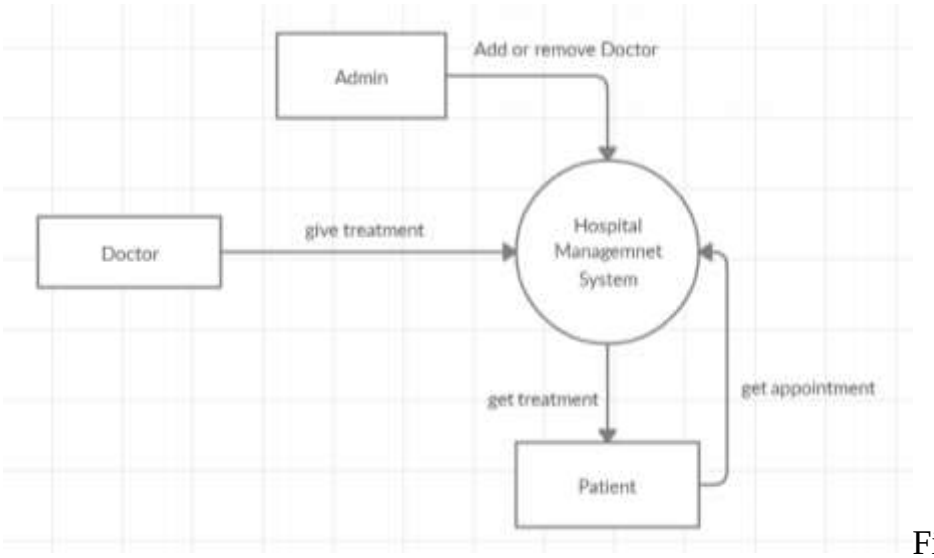

Fig. 12.2

Level 1 :

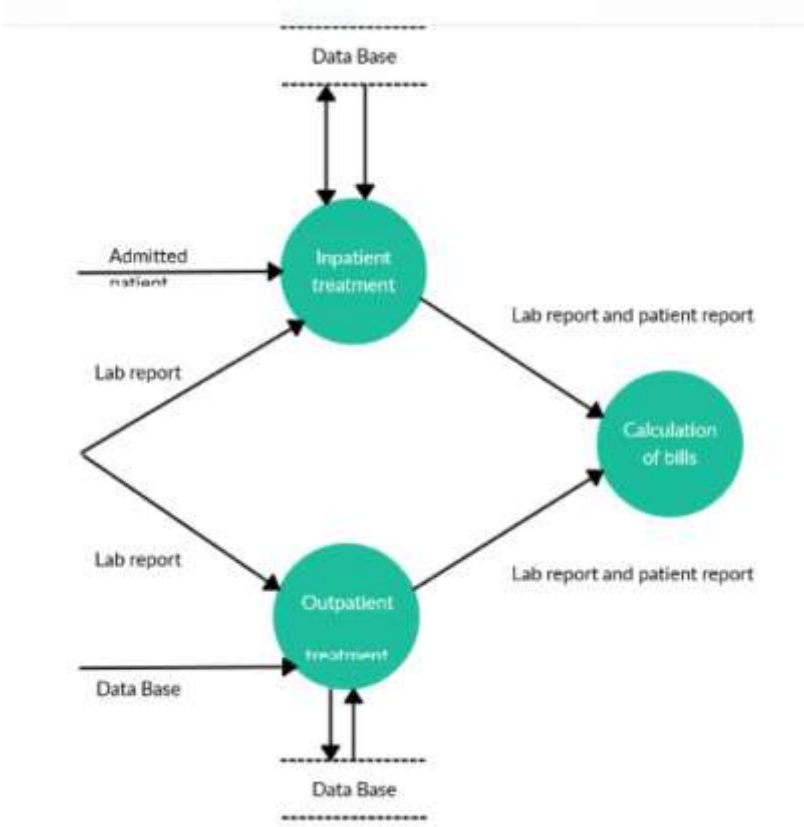

Fig. 12.3

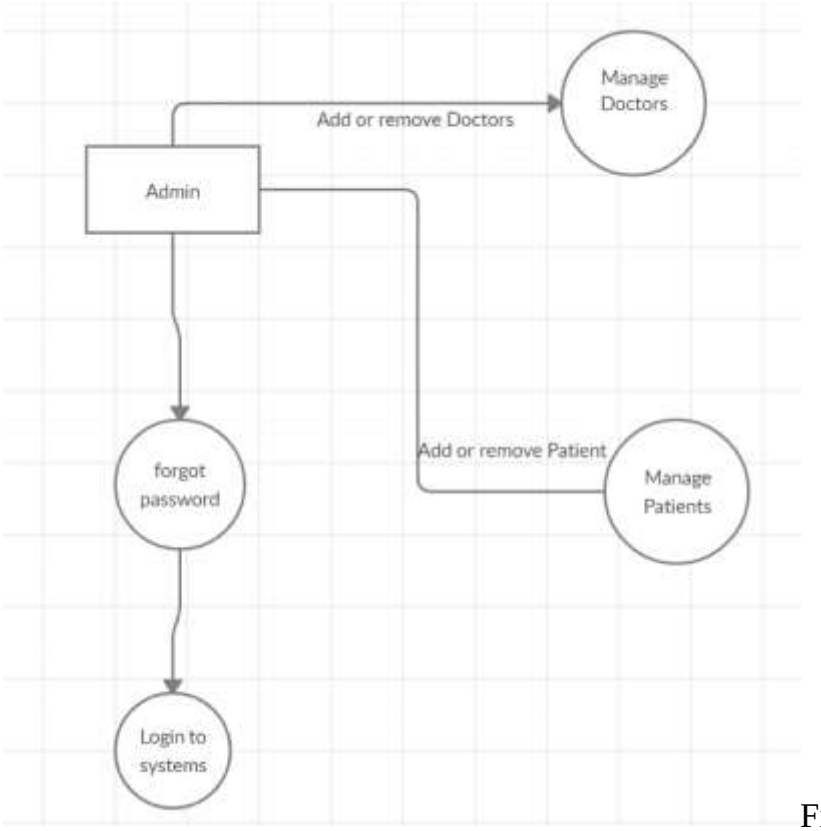

Fig. 12.4

Level 2 :

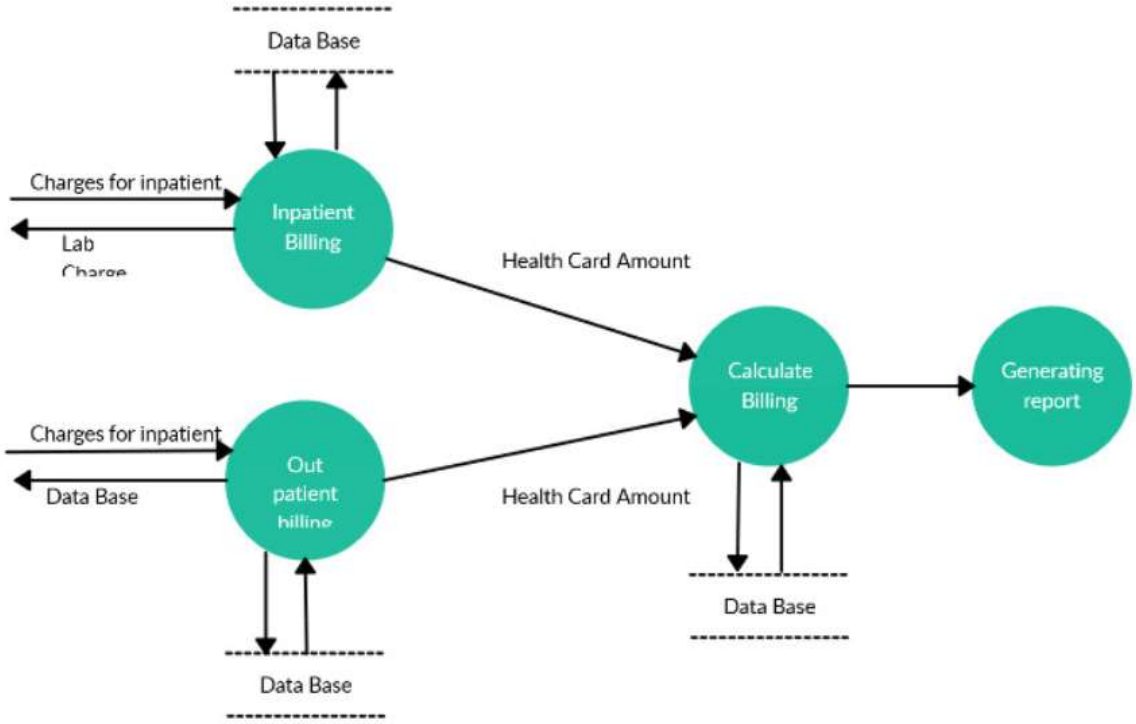

Fig. 12.5

13. COCOMO MODEL

Cocomo (Constructive Cost Model) is a regression model based on LOC, i.e **number of Lines of Code**. It is a procedural cost estimate model for software projects and often used as a process of reliably predicting the various parameters associated with making a project such as size, effort, cost, time and quality.
The key parameters which define the quality of any software products, which are also an outcome of the Cocomo are primarily Effort & Schedule:

- **Effort:** Amount of labor that will be required to complete a task. It is measured in person-months units.
- **Schedule:** Simply means the amount of time required for the completion of the job, which is, of course, proportional to the effort put. It is measured in the units of time such as weeks, months.

Different models of Cocomo have been proposed to predict the cost estimation at different levels, based on the amount of accuracy and correctness required. These models pertaining to different system types are mentioned below.

Definition of organic, semidetached, and embedded systems:

1. **Organic –** A software project is said to be an organic type if the team size required is adequately small, the problem is well understood and has been solved in the past and also the team members have a nominal experience regarding the problem.
2. **Semi-detached –** A software project is said to be a Semi-detached type if the vital characteristics such as team-size, experience, knowledge of the various programming environment lie in between that of organic and Embedded. The projects classified as Semi-Detached are comparatively less familiar and difficult to develop compared to the organic ones and require more experience and better guidance and creativity. Eg: Compilers or different Embedded Systems can be considered of Semi-Detached type.
3. **Embedded –** A software project with requiring the highest level of complexity, creativity, and experience requirement fall under this category. Such software requires a larger team size than the other two models and also the developers need to be sufficiently experienced and creative to develop such complex models.

All the above system types utilize different values of the constants used in Effort Calculations.

Types of Models: COCOMO consists of a hierarchy of three increasingly detailed and accurate forms. Any of the three forms can be adopted according to our requirements. These are types of COCOMO model:

1. Basic COCOMO Model
2. Intermediate COCOMO Model
3. Detailed COCOMO Model

The first level, **Basic COCOMO** can be used for quick and slightly rough calculations of Software Costs. Its accuracy is somewhat restricted due to the absence of sufficient factor considerations.

Intermediate COCOMO takes these Cost Drivers into account and **Detailed COCOMO** additionally accounts for the influence of individual project phases, i.e in case of Detailed it accounts for both these cost drivers and also calculations are performed phase wise henceforth producing a more accurate result. These two models are further discussed below.

Estimation of Effort: Calculations

1. **Basic Model**

   $$E=a(KLOC)^b$$

   The above formula is used for the cost estimation of for the basic COCOMO model, and also is used in the subsequent models.The effort is measured in Person-Months and as evident from the formula is dependent on Kilo-Lines of code. These formulas are used as such in the Basic Model calculations, as not much consideration of different factors such as reliability, expertise is taken into account, henceforth the estimate is rough.

2. **Intermediate Model**

   The basic Cocomo model assumes that the effort is only a function of the number of lines of code and some constants evaluated according to the different software system. However, in reality, no system's effort and schedule can be solely calculated on the basis of Lines of Code. For that, various other factors such as reliability, experience, Capability. These factors are known as Cost Drivers and the Intermediate Model utilizes 15 such drivers for cost estimation.

   Classification of Cost Drivers and their attributes:

   (i) Product attributes –
   - Required software reliability extent
   - Size of the application database
   - The complexity of the product

   (ii) Hardware attributes –
   - Run-time performance constraints
   - Memory constraints
   - The volatility of the virtual machine environment
   - Required turnabout time

   (iii) Personnel attributes –
   - Analyst capability
   - Software engineering capability

- Applications experience
- Virtual machine experience
- Programming language experience

(iv) Project attributes –

- Use of software tools
- Application of software engineering methods
- Required development schedule

- The project manager is to rate these 15 different parameters for a particular project on a scale of one to three. Then, depending on these ratings, appropriate cost driver values are taken from the above table. These 15 values are then multiplied to calculate the EAF (Effort Adjustment Factor). The Intermediate COCOMO formula now takes the form:

$$E=(a(KLOC)^b)*EAF$$

3. **Detailed Model**

Detailed COCOMO incorporates all characteristics of the intermediate version with an assessment of the cost driver's impact on each step of the software engineering process. The detailed model uses different effort multipliers for each cost driver attribute. In detailed cocomo, the whole software is divided into different modules and then we apply COCOMO in different modules to estimate effort and then sum the effort.

The Six phases of detailed COCOMO are:

1. Planning and requirements
2. System design
3. Detailed design
4. Module code and test
5. Integration and test
6. Cost Constructive model

The effort is calculated as a function of program size and a set of cost drivers are given according to each phase of the software lifecycle.

The solution of the given situation is as given below:

This solution described and used the intermediate model with organic type.

In the given condition we have two pools

① First Pool : highly capable and less Experienced in language

② Second Pool : Low Capable and Experienced in language used.

given Kilo lines of code = 200

The model used is intermediate and of organic type.

1. Efforts:

For first pool: $E_1 = (a(KLOC)^b) * EAF$

$= (3.2)(200)^{1.05} * (0.86)(1.07)$

$= 767.561$ person_month

For Second Pool: $E_2 = (a(KLOC)^b) * EAF$

$= (3.2)(200)^{1.05} * (1.17)(0.95)$

$= 927.129$ person_month

2. Development time:

First Pool: $D_1 = C(Effort)^d$

$= (2.5)(767.561)^{0.38} = 31.209$ months

Second Pool: $D_2 = C(Effort)^d$

$= (2.5)(927.129)^{0.38} = 33.531$ months

3. ~~Time~~ Average Person Required:

First Pool: $P_1 = \frac{Efforts}{Development\ time} = \frac{767.561}{31.209} = 24.594$

≈ 25 persons

Second Pool: $P_2 = \frac{927.129}{33.531} = 27.64$

≈ 28 Persons

So the value used for constants a and b in calculating Efforts are 3.2 and 1.05.

The value used for constants c and d in calculating Development Time are 2.5 and 0.38.

Hence, we can say that first pool is more efficient than second in each aspect. So, the cost involved in the first is less than the cost involved in second.

14. GitHub

GitHub is a code hosting platform for collaboration and version control. GitHub lets you (and others) work together on projects.

GitHub is a Git repository hosting service, but it adds many of its own features. While Git is a command line tool, GitHub provides a Web-based graphical interface. It also provides access control and several collaboration features, such as a wikis and basic task management tools for every project.

The flagship functionality of GitHub is “forking” – copying a repository from one user’s account to another. This enables you to take a project that you don’t have write access to and modify it under your own account. If you make changes you’d like to share, you can send a notification called a “pull request” to the original owner. That user can then, with a click of a button, merge the changes found in your repo with the original repo.

These three features – fork, pull request and merge – are what make GitHub so powerful. Gregg Pollack of Code School (which just launched a class called TryGit) explains that before GitHub, if you wanted to contribute to an open source project you had to manually download the project’s source code, make your changes locally, create a list of changes called a “patch” and then e-mail the patch to the project’s maintainer. The maintainer would then have to evaluate this patch, possibly sent by a total stranger, and decide whether to merge the changes.

This is where the network effect starts to play a role in GitHub, Pollack explains. When you submit a pull request, the project’s maintainer can see your profile, which includes all of your contributions on GitHub. If your patch is accepted, you get credit on the original site, and it shows up in your profile. It’s like a resume that helps the maintainer determine your reputation. The more people and projects on GitHub, the better idea picture a project maintainer can get of potential contributors. Patches can also be publicly discussed.

Even for maintainers who don’t end up using the GitHub interface, GitHub can make contribution management easier. “I end up just downloading the patch anyway, or merging from the command line instead of from the merge button,” says Isaac Schlueter, the maintainer of the open source development platform Node.js. “But GitHub provides a centralized place where people can discuss the patch.”

Besides its public facing open source repositories, GitHub also sells private repositories and on-premise instances of its software for enterprises. These solutions obviously can’t take full advantage of GitHub’s network effect, but they can take advantage of the collaboration features.

GitHub is so intuitive to use and its version-control tools are so useful for collaboration, nonprogrammers have also begun to use GitHub to work on document-based and multimedia projects. GitLab is an open source alternative to GitHub.

GitHub offers an on-premises version in addition to the well-known SaaS product. GitHub Enterprise supports integrated development environments and continuous integration tool integration, as well as a litany of third-party apps and services. It offers increased security and auditability than the SaaS version.

15. BitBucket

Bitbucket is our Git repository management solution designed for professional teams. It gives you a central place to manage git repositories, collaborate on your source code and guide you through the development flow. It provides awesome features that include:

- Access control to restrict access to your source code.
- Workflow control to enforce a project or team workflow.
- Pull requests with in-line commenting for collaboration on code review.
- Jira integration for full development traceability.
- Full Rest API to build features custom to your workflow if they are not already available from our Marketplace

Bitbucket Cloud (previously known as Bitbucket) is written in Python using the Django web framework. Bitbucket Cloud is hosted on Atlassian's servers and accessed via a URL. Bitbucket Cloud has an exclusive built-in continuous integration tool, Pipelines, that enables you to build, test and deploy from directly within Bitbucket. However, there are some restricted functions in Atlassian Cloud apps.

Bitbucket Server is hosted on-premise, in your environment. Bitbucket Server does not come with a built-in testing and deployment tool, but it has strong integrations with Bamboo, our powerful continuous integration and continuous delivery tool that allows you to completely automate your build processes. There are also more apps available than for Cloud, and the license is perpetual.

Bitbucket Data Center (our Enterprise offering) looks like a single instance of Bitbucket Server to users, but it is hosted on a number of servers in a cluster on your environment. This provides important benefits over Bitbucket Server:

- Performance at scale: A cluster of many machines running Bitbucket Server can handle more load than a single machine.
- High Availability: If one cluster node goes down, then the remaining cluster node(s) can continue servicing requests so users should see little or no loss of availability.
- Smart mirroring: Smart Mirroring can improve Git clone speeds for distributed teams working with large repositories.

Scope

Bitbucket is mostly used for code and code review. Bitbucket supports the following features:

- Pull requests with code review and comments
- Bitbucket Pipelines
- 2 step verification and required two step verification
- IP whitelisting
- Merge Checks
- Code search (Alpha)
- Git Large File Storage (LFS)
- Documentation, including automatically rendered README files in a variety of Markdown-like file formats
- Issue tracking
- Wikis
- Static sites hosted on Bitbucket Cloud: Static websites have the bitbucket.io domain in their URL
- Add-ons and integrations
- REST APIs to build third party applications which can use any development language
- Snippets that allow developers to share code segments or files
- Smart Mirroring