Unit 3 – Chapter 8

Software Design and Development

Prerequisites:

Software design fundamentals, object-oriented programming concepts

Unit Outcomes:

The objective of this part of the unit is to understand the process of software design and development. At the end of this chapter, the student will be able to:

- Understand object-oriented software design and its activities
- Understand how to use unified modeling language (UML) diagrams for object-oriented design (OOD)
- Analysis of interaction and behavioral diagrams
- Discuss architectural software design and architecture patterns: client-server, repository and pipe and filter

8.1 Introduction to OOD

The important parts of the OOD include objects, classes, messages, concepts such as abstraction, encapsulation, inheritance, and polymorphism. These concepts are outlined in brief as follows:

1. Class is an abstract data type that combines functions and data in one unit. Objects are abstractions of the real world, and they consist of both state and operations. The state is represented as a set of object attributes. The operations or functions used in one object provide services to other objects. Object communication uses message passing and eliminating duplicate or shared data. Objects are created according to some class definition. Thus, a class definition serves as a template for objects.
2. Abstraction is a concept where the important properties are highlighted, and unimportant or irrelevant information is removed.

3. Encapsulation combines data and functions in one unit. It also includes data hiding where the access is restricted to the data and functions from the outside world.
4. Inheritance in OOD allows reusability, where one or more classes can be generated using existing data and functions/modules in different classes.
5. Polymorphism: OOD languages provide a mechanism where the data can be processed in more than one form. The object can take many forms to perform different jobs.

The OOD design process takes place in the following steps:

a) Define the context of the use of the system

b) Design the system architecture

c) Identify the important/principal system objects

d) Develop design models

e) Specify object interfaces

8.1.1The Unified Modelling Language (UML) in OOD

Many distinct notations for describing the OOD process were proposed in the 1980s and 1990s. The UML is an integration of these notations that describes different design models that are developed during OO analysis and design. It is now a *de facto* standard for OO modeling. (*de facto* means the standard that is commonly used in the market). The notations in UML are divided into three categories: Things, relationships, and diagrams. Things include notations for class, object, interface, use-case including actor, package, a component in structural form. Relationships are used in showing the behavior, the different notations include start and stop state, action box, decision box, annotation (explanation), etc. The relationships in UML are association relationship, dependency relationship, generalization relationship, and realization relationships. The different diagrams are used to show the structure, behavior, and interaction in a system

There are various system models to represent them from different perspectives. Some of the commonly used diagrams are:

- To model the context or environment of the system is an external perspective.

- Interaction perspective is used for showing the behavior of the system. These diagrams capture the structure of a system, the flow of messages between objects, and the interaction between them.
- The behavioral perspective where the dynamic behavior of the system is modeled.
- The UML diagrams make use of different notations and the relationship notations. Some of the important UML diagrams in the context of OOD are:

  a) Structural diagrams: Class diagrams, which show the object classes in the system and the associations between these classes.

  b) Interaction diagrams: These diagrams show interaction, for example, use case diagrams, show the interactions between a system and its environment, and sequence diagrams, which show interactions between actors and the system and between system components.

  c) Behavioural diagrams: Activity diagrams, which show the activities involved in a processor data. Also, sequence diagrams, show how the exchange of messages between the objects.

8.2 Context models

At the beginning of the software design, it is necessary to set boundaries of a system to be developed. The context models are used to show the scope of the system to be designed with the environment. They are used to show which functionalities are in the system and which functions are beyond the system's boundary. Context models are the very starting point of the design phase of the software.

For example, in ATM context diagram, customer, bank, pin, balance, amount to be withdrawn, etc. are relevant, other details such as customer's PAN number, address, loan, insurance is used in other systems of a bank, but they are not suitable in ATM context. Simple context models are used along with other models such as the process model. A simple context diagram for the ATM System is shown in Figure 8.1.

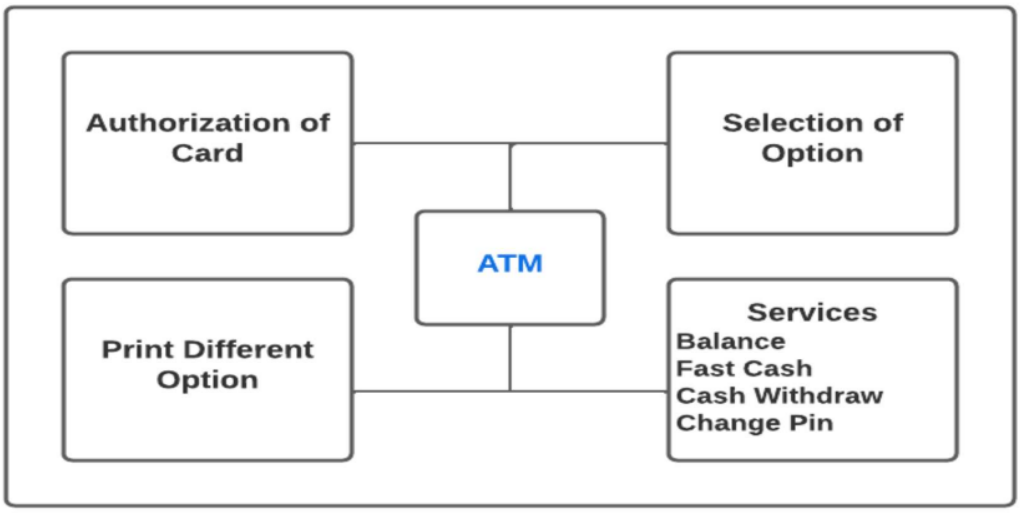

Figure 8.1 Context Model of ATM

The different operations identified in the entities of context models are shown using process models. Process models reveal how the system being developed is used in broader business processes. It gives details of how the system works.

8.3 Interaction models

All the systems have some interaction with user input and output, with other components and systems. The interaction helps in understanding what are the requirements in a system. To show the behavior of the system, interaction models are used. These models capture the structure of a system, the flow of messages between objects, and the interaction between them. Some of the commonly used interaction models in UML are:

1. Use case modeling where the interaction of how the system works with actors (users or another system) and test cases is shown.
2. Sequence diagrams: Shows the interaction of messages amongst classes.

8.3.1 Use case diagram:

Use diagram represents a sequence of actions with users referred to as actors. Actors are shown as stick figures. The communication between actor and use case is shown by solid lines. Use case diagram is considered as the basic diagram that is useful for gathering requirements of a new system to be formed. Use cases represent the functional requirements of a system. Use case diagrams help in expressing the context of a system

and the different test cases to be used. A simple use case diagram for the ATM system is shown in Figure 8.2. In this case, there are two actors, a cardholder or client, and a bank account for managing the client's data. Actors may be people, devices, or other systems.

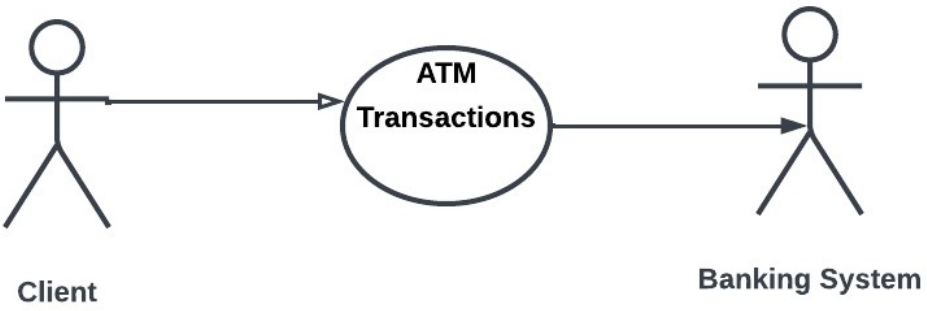

Figure 8.2 Use case diagram showing the relationship between a client and banking using ATM.

As the direction of the arrow here, when a client asks for the data and the banking record system returns it. The ATM system can involve many use cases as shown in Figure 8.3. Here the banking in charge person is managing several operations.

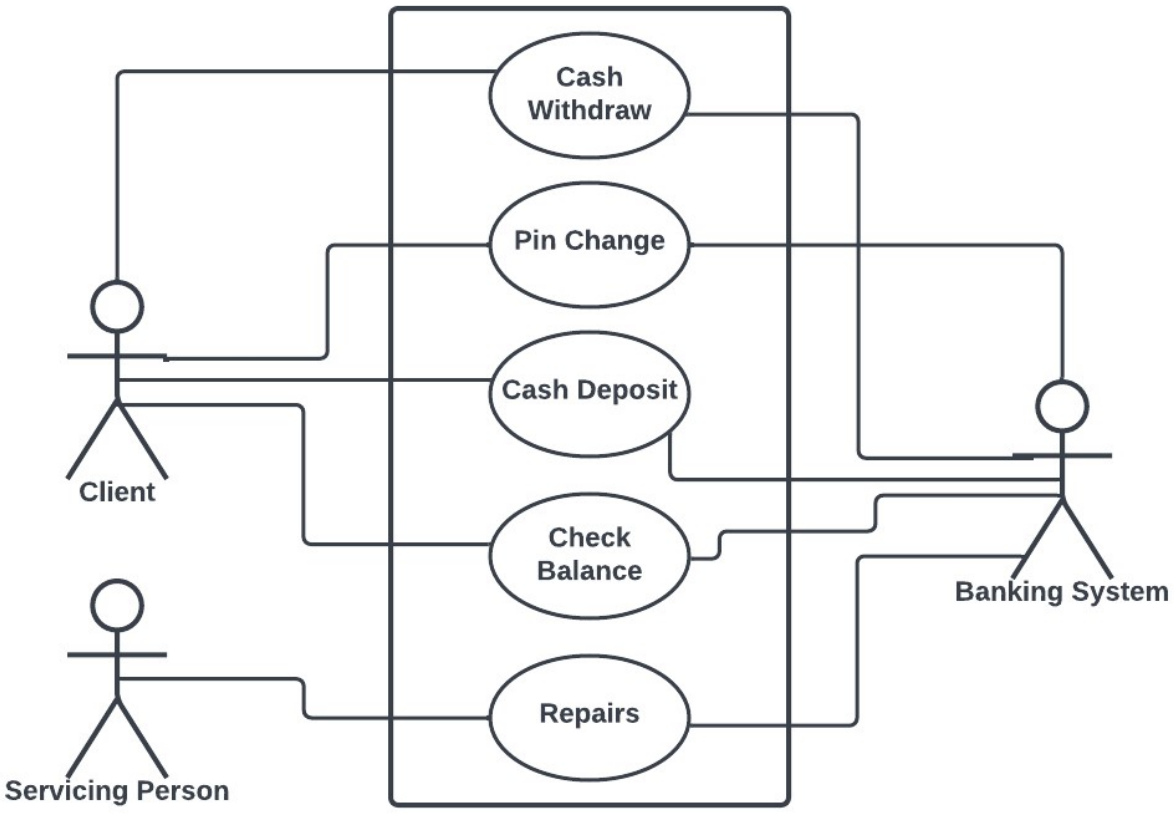

Figure 8.3 Use Case Diagram for ATM

8.3.2 Sequence Diagram

These diagrams model the communication of objects based on the time sequence. For a particular scenario, how the objects interact with others is shown using sequence diagrams. A sequence diagram, where the interaction of a bank person, user, and admin to check balance is shown in Figure 8.4

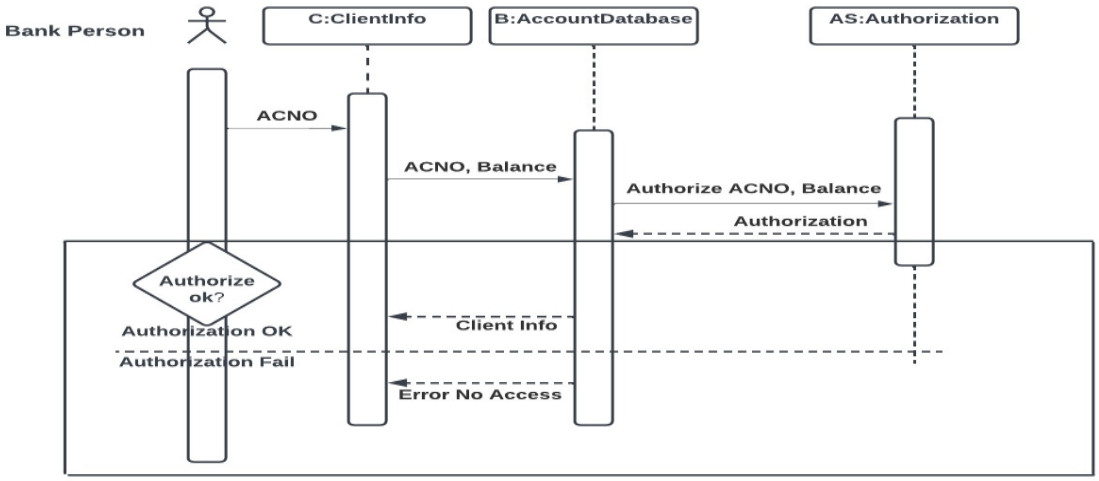

Figure 8.4 Sequence diagram for bank account authorization

A sequence diagram for checking patients' records is shown in Figure 8.5.

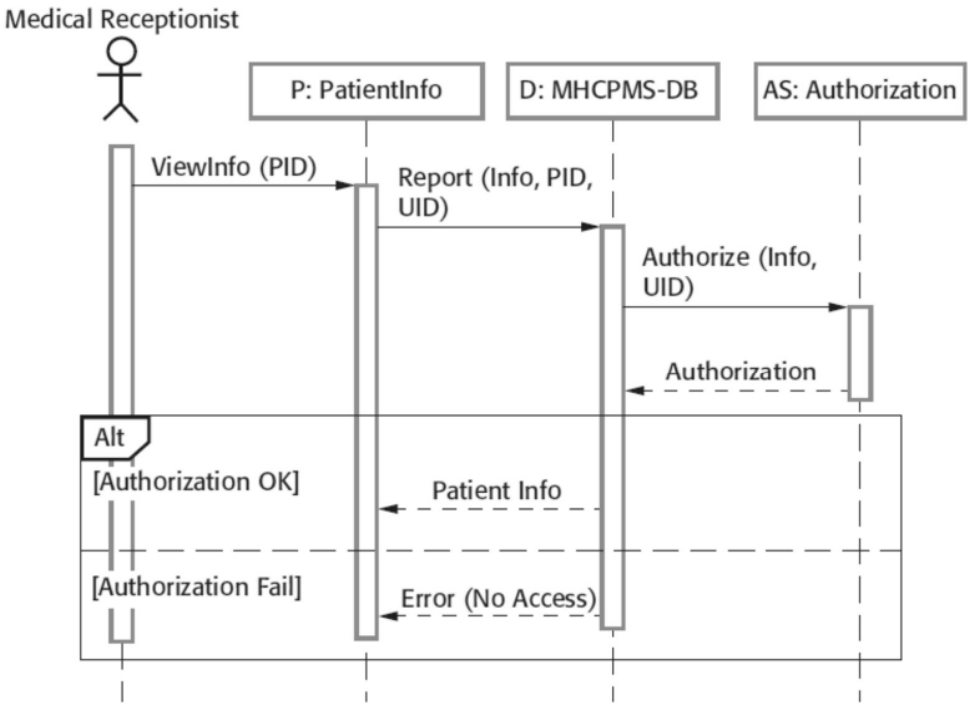

Figure 8.5: Sequence Diagram for checking patient information

Here the objects and the time at which they are called are placed at the top in a rectangle box with a dotted vertical line. The arrows with labels show the interaction between the objects. The conditions such as is the authorization OK, or FAIL is shown in the square bracket with a box ALT. The steps in the above sequence diagrams are as follows:

1. The medical receptionist asks for the information of the patient with the patient's identifier (PID), the method (function) used is ViewInfo. P is an instance that displays a patient's information.
2. The instance P calls the database with PID and checks whether the user is authorized or not. It is checked in the database (MHCPMS-DB). Here UID indicates the number given to the medical receptionist.
3. The MHCPMS-DB calls Authorization object AS: Authorization to check whether the patient record is authorized. If yes, then a form with patients' information is returned. If No, then an error message is displayed.

8.4 Structured Models

Structured models display the organization of a system indicating the components and the relationships among the components. Structural models may be static or dynamic Static models represent the structure of a system design. The dynamic models show the system organization during execution.

A structured diagram depicts the different structures or concepts used in software development such as process, records, database, class, package, etc. Structured diagrams also show the relation between these structures. The most common example of a structured diagram is the class diagram.

8.4.1 Class Diagrams

The class diagram shows the classes with attributes, functions, and inter-relationships.

The class diagram can be used at different stages in software design. At the initial stage, it may represent the important objects of the system. It is simply given by writing a class name in the box. The Association of one class can also be identified at this stage. A simple class diagram indicating patient record and patient is shown in Figure 8.6.

Figure 8.6: Association of UML Class

Here the association is 1:1 and it is indicated on the line connecting two classes. There can be multiple objects connected. If the exact number is known, then it is mentioned or (*) is shown. In Figure 8.7, there can be many subjects assigned to the student. This is shown in the using 1.*, whereas only one teacher is assigned to a student.

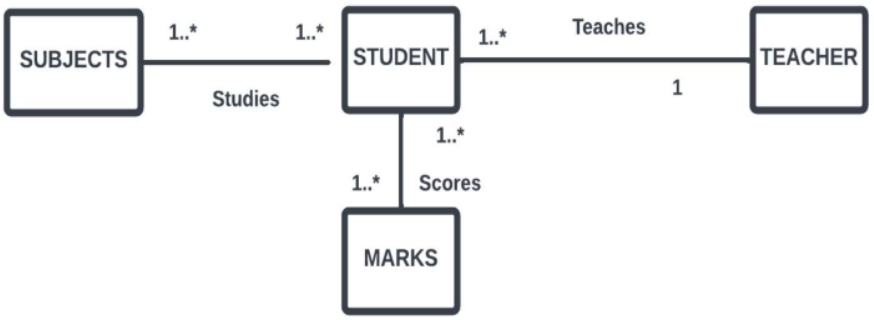

Figure 8.7: Multiple Association of UML Class

A class diagram for student management is shown in Figure 8.8. In the class diagram, the name is written in the top section. The class attributes are written in the middle section. It includes attribute names and types. The functions are written in the lower section of the rectangle. For simplicity, only a few attributes and operations are considered.

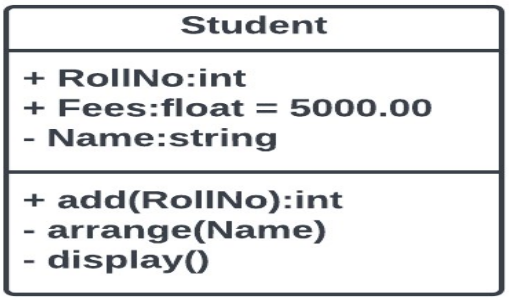

Figure 8.8: Class Diagram for student enrolment

The two important relationships used in class diagrams are as given below.

Generalization:

This is a relationship used in OOD to represent complexity. Instead of showing detailed information of every entity, here many general classes are used. An example of generalization in a hospital is shown in the following Figure 8.9.

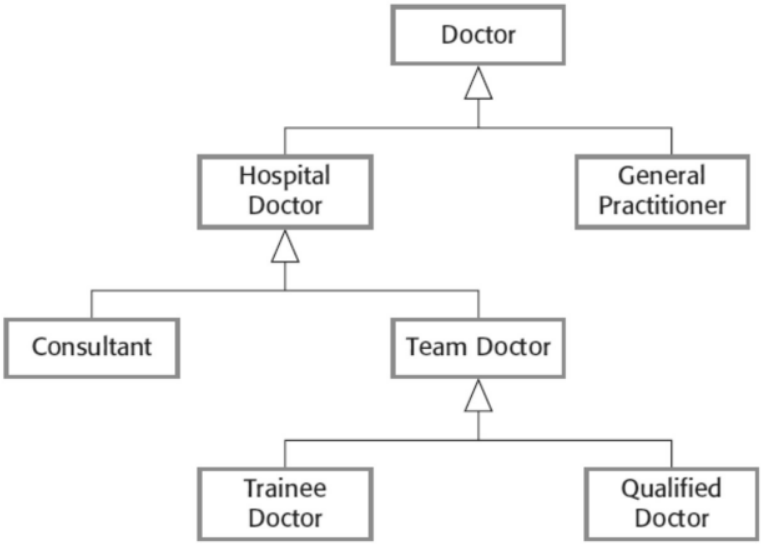

Figure 8.9: Generalization of Hierarchy of doctors

As shown in the above diagram, the two categories of doctors namely Hospital and General practitioners are generalized as a doctor. This association is shown by a specific symbol in UML. The class team doctor consists of a Trainee doctor and a Qualified doctor. The recently passed medical students are shown in as Trainee doctor and he/she is supervised by the Team doctor.

Generalization/Inheritance: Generalisation in the UML is implemented as an inheritance in OO programming languages. Generalization is a bottom-up process that is used to build single or more general classes that are similar. Inheritance allows reusability. Objects are members of classes that define attribute types and operations. Classes may be arranged in a class hierarchy where one class (a super-class) is a generalization of one or more other classes (sub-classes). sub-class inherits the attributes and operations from its superclass and may add new methods or attributes of its own. For example, if there are two classes *Library* and *Hostel in a college-based software,* then these classes can reuse the members of *the Admission* class, where all the data related to the students is stored. Similarly, a *Hospital Doctor* and *General Practitioner* can use some of the

properties of a *Doctor* class. Generalization with class attributes is shown in Figure 8.10. The classes *Hospital Doctor* and *General Practitioner* can use the data members *Name, Phone #, Email* and *Resister (), De-Register ()* methods in addition to their members.

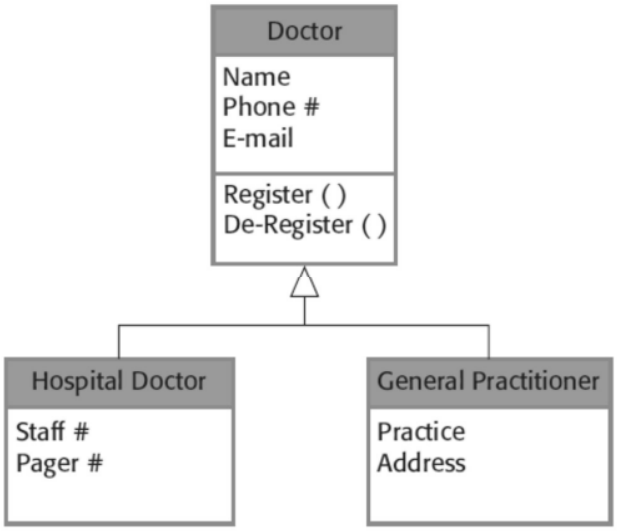

Figure 8.10: Inheritance amongst classes of doctors

Another example of generalization for a bank account is given below.

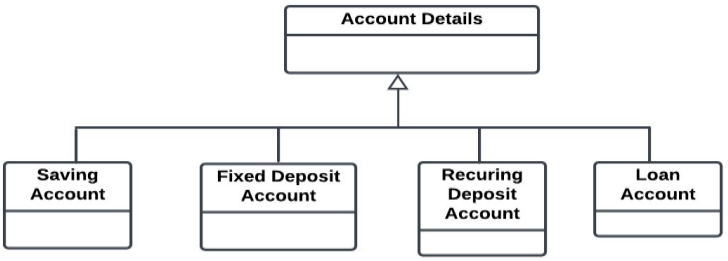

Figure 8.11: Inheritance amongst classes of accounts for a user

Aggregation: Aggregation denotes the combination of more than one object that is configured together. An example of aggregation is shown in figure 8.12. Here the children of one class are independent of the other. In the following diagram, if a teacher resigns or a student quits the college, then that specific class can be removed.

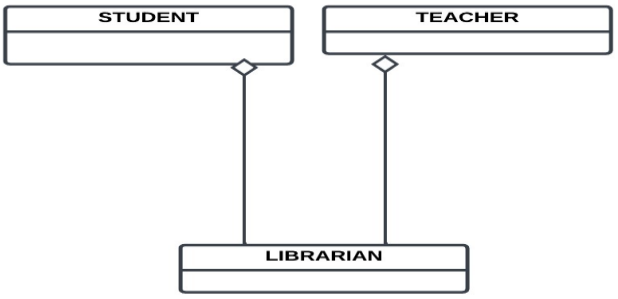

Figure 8.12: Aggregation of classes

Composition: Composition implies one class containing another. The contained class cannot exist independently. For example, a library cannot exist without Books as shown in Figure 8.13.

Figure 8.13 Composition of classes

8.5 Behavioral diagrams

When the system is in execution then its dynamic behavior can be shown using behavioral models. The system responds to the data or some event from the other systems. For example, when a telephone system will take the customer's calls and internet usage as input and generate the bill. This example shows how the system responds to data. Consider a landline telephone switching office where the system responds to events such as "dial a number", "keep the receiver on the hook". This is an example of an event-driven system. Let us discuss data-driven and event-driven modeling using some examples.

8.5.1 Data-driven Modelling

Data-driven modeling can be performed using an activity diagram in UML.

Activity Diagrams:

Activity means operation. This diagram shows the flow of control from activity to activity. Activity diagrams indicate activity using circle, rectangle, and arrow notations. The filled circles show the start and stop of the activity is shown using a filled circle. The activities are shown using rounded rectangles. Arrows are used to show the data flow, if there are

conditions then annotations are used on arrows to indicate that. The diamond represents the decision box. The process of reserving tickets for a specific class is shown in Figure 8.14.

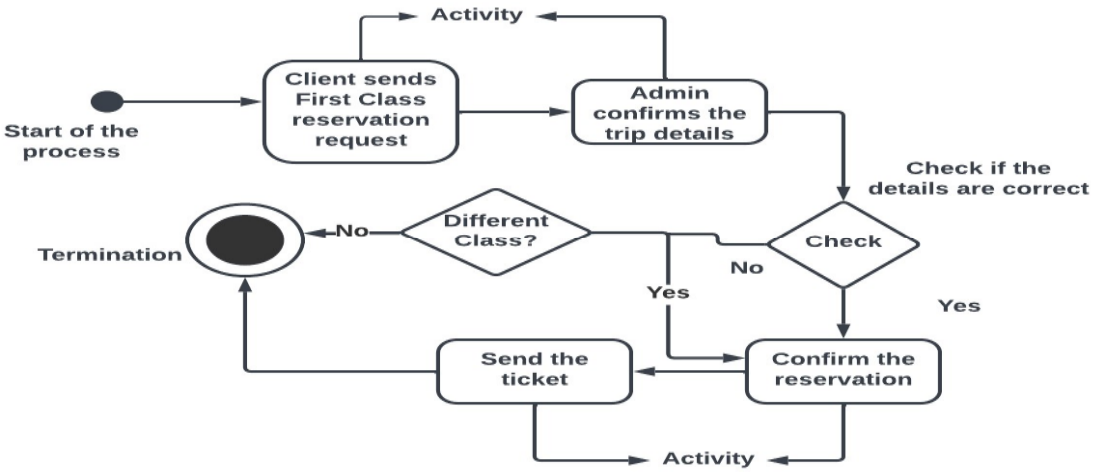

Figure 8.14 Activity Diagram for reservation of tickets

8.5.2 Event-driven Modelling

Event is termed as stimuli. Event-driven modeling is the response of a system to external and internal events. Event-driven models are represented using state diagrams in UML In these diagrams, how a single object responds to different events in its lifetime is shown. It captures different states through which an object is going through. A state diagram for a microwave oven is shown below. Rounded rectangles represent the states. A transition from one state to another is shown by a line with an arrowhead. Transitions may be self-transition returning to itself. States can be compounds where one state can point to another state. The event-driven modeling for the microwave oven example is shown in Figure 8.15. The sequence of activities in a microwave oven can be given as:

1. Choose the half power or full power level.
2. Use the keypad and enter the time
3. Press Start with which the food item gets cooked intime.

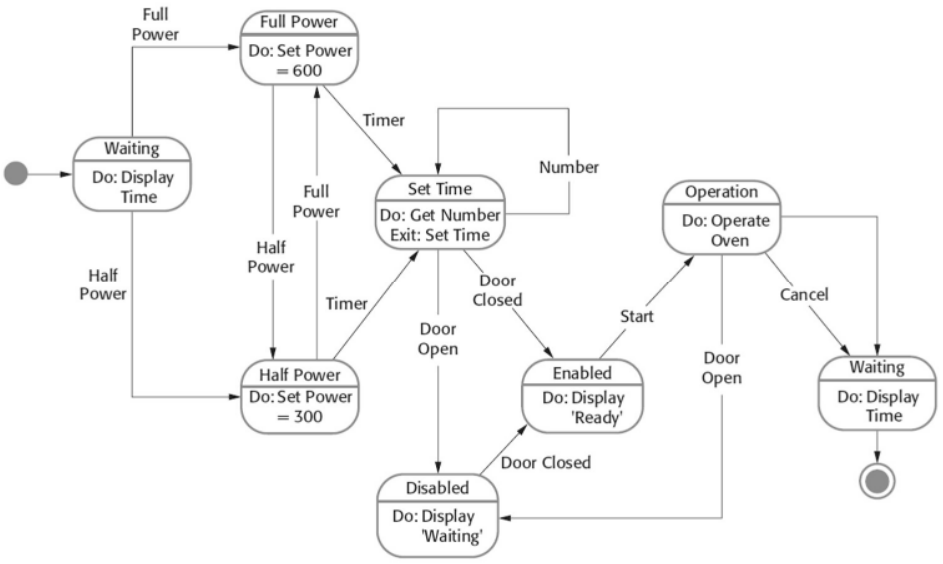

Figure 8.15: State Diagram for Microwave Oven Operations

The states and the stimuli for the above-shown example are given in figure 8.16.

State	Description
Waiting	The oven is waiting for input. The display shows the current time.
Half power	The oven power is set to 300 watts. The display shows 'Half power'.
Full power	The oven power is set to 600 watts. The display shows 'Full power'.
Set time	The cooking time is set to the user's input value. The display shows the cooking time selected and is updated as the time is set.
Disabled	Oven operation is disabled for safety. Interior oven light is on. Display shows 'Not ready'.
Enabled	Oven operation is enabled. Interior oven light is off. Display shows 'Ready to cook'.
Operation	Oven in operation. Interior oven light is on. Display shows the timer countdown. On completion of cooking, the buzzer is sounded for five seconds. Oven light is on. Display shows 'Cooking complete' while buzzer is sounding.
Stimulus	**Description**
Half power	The user has pressed the half-power button.
Full power	The user has pressed the full-power button.
Timer	The user has pressed one of the timer buttons.
Number	The user has pressed a numeric key.
Door open	The oven door switch is not closed.
Door closed	The oven door switch is closed.
Start	The user has pressed the Start button.
Cancel	The user has pressed the Cancel button.

Figure 8.16: The state and stimuli diagram for Microwave Oven

8.6 Introduction to Architectural Design

The organization and design of the overall structure of a system are described in architectural design. It is the starting stage in the software design process. The important link between the requirements and design is identified using architectural design. The different relationships between software components as we studied in OOD are identified in this stage. An architectural model with a set of communicating components is an output of architectural design. It serves as a blueprint of a system.

One of the commonly used methods for architectural design is the agile process. In this method, the requirements and solutions get evolved. At the early stage, the focus is on designing system architecture. Incremental development of architectures is not usually successful. Designing a software architecture is crucial as the performance and maintenance of a system get affected by the design. The non-functional requirements are given at most importance is given in the system's architecture.

Advantages of architectural design are:

1. **Stakeholder communication:** The people involved in the system get a high-level overview of the system to be developed.
2. **System analysis:** The system architecture checks whether the design of a system is meeting important requirements such as performance, maintainability, and reliability. Some system analysis is performed at the early stage of architectural design.
3. **Large-scale reuse:** An architectural model is a compact model of a system, and it can be helpful in software reuse. Support to large-scale software reuse can be given using an architectural design model.

Architectural design is related to many non-functional requirements such as performance, safety, security, availability, and maintainability. All the relevant information in the system cannot be represented using a single architectural model. Thus, one view or perspective of the system can be given using architectural design. The system decomposition into modules, their behavior during run time and across the network can be shown using architectural view.

8.7 Architectural Views

Usually, multiple views of the software architecture are presented. There are four fundamental architectural views, which can be linked through common use cases or scenarios.

1. A **logical view** depicts how the system requirements are related to the entities in system architecture. The key abstractions such as objects or object classes are identified in this view.
2. A **process view** represents how the different processes interact during runtime. The non-functional system characteristics such as performance and availability can be judged in this view.

3. A **development view** shows how the software in the development stage is decomposed into components. Each component can be implemented by a single developer or development team. The software project managers and programmers use this view.
4. A **physical view** is used by the systems engineer for checking how system hardware and software components are distributed across the processors in the system. This view is useful in the deployment of the project. Installation of a developed system across computer networks is yet another application of physical view.

8.8 Architectural Patterns

The architecture of a software system may be based on a particular architectural pattern or style. Architectural patterns give an abstract description of a system organization. It helps present, share, and reuse knowledge about software systems. An Architectural pattern should describe a system organization that was used in previous systems. It should include information on when it is and is not appropriate to use that pattern and details on the pattern's strengths and weaknesses. Some of the commonly used architectural patterns are discussed in this section are:

1. Layered Architecture
2. Repository Architecture
3. Client-Server Architecture
4. Pipe-filter Architecture

1) Layered architecture

Layered architecture is the most common architecture pattern in software development. It allows separation and independence in the design. This means that layers can be modified without affecting other layers. Independence allows each layer to perform a single task. With these features, it is possible to make local changes in the software. Here, the system functionality is organized into separate layers, and each layer only relies on the facilities and services offered by the layer immediately beneath it. The characteristic of the layered approach is as given in Table 8.1.

Name	Layered architecture
Description	The details about the organization of the system into layers, with related functions associated with each layer, are given.
Example	OSI Network model. Gmail layered model
When used	When the system development is distributed across the teams and each team is responsible for a layer of functionality. It is used when we have to build new facilities on top of existing systems.
Advantages	A layered approach is simple and easy to implement. The entire layer can be replaced as there is less dependency. Testing and maintenance is easier because of separate layered functions
Disadvantages	In practice, it is very difficult to make a clear separation between layers. Scalability and parallel processing are difficult in a layered approach.

Table 8.1: Layered Architecture

With the help of the layered approach, the system can be developed using an incremental method. As a layer is developed, some of the services provided by that layer may be made available to users. The architecture is also changeable and portable. If its interface is unchanged, a new layer with extended functionality can replace an existing layer without changing other parts of the system. Furthermore, when layer interfaces change or new facilities are added to a layer, only the adjacent layer is affected.

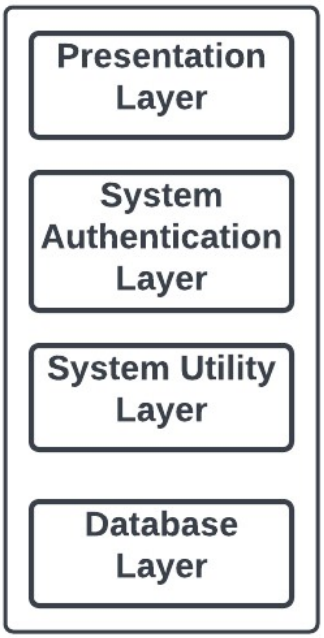

Fig 8.17: Layered Architecture

Fig 8.17 is an example of a layered architecture with four layers. The lowest layer includes system support database support. The next layer is the system utility layer which includes components related to different functions in the application. The third layer is concerned with providing user authentication and authorization. The topmost layer is the presentation layer giving facilities for the user interface. The number of layers in software design can be arbitrary. Any of the layers could be split into two or more layers.

2) Repository architecture

The repository pattern describes how a set of interacting components can share data. Most systems that use large amounts of data are organized around a shared database or repository. This pattern is described in Table 8.2. This pattern is therefore suited to applications in which data is generated by one component and used by another. Examples of this type of system include command and control systems, management information systems, Computer-Aided Design (CAD) systems, and interactive development environments for software. The centralized data can be stored in sub-systems for its components.

Name	Repository
Description	All data in a system is saved in a central repository. It is accessible to all system components. Components interact only through the repository.
Example	An IDE where the components use a repository of system design information. Each software tool generates information, which is then available for use by other tools.
When used	When there is a system with voluminous data, it is necessary to store that information for a long time. It is also used in data-driven systems where the inclusion of data in the repository triggers an action or tool.
Advantages	All data can be managed consistently (e.g., backups done at the same time) as it is all in one place. An efficient way of sharing data
Disadvantages	The failure of the single point used as a repository may shut down the whole system. If the communication is only through the centralized approach, then the distribution across many computers can be complex and difficult.

Table 8.2: Repository Architecture

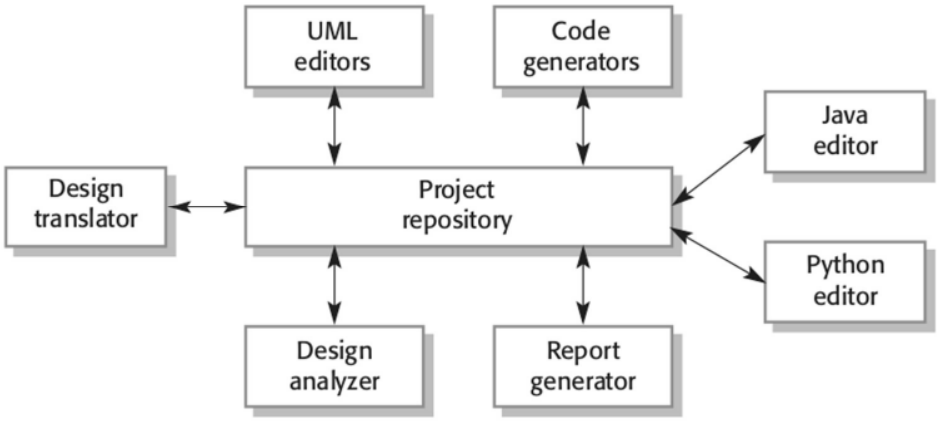

Fig 8.18: IDE with Repository Architecture

Figure 8.18 illustrates a situation in which a repository might be used. This diagram

shows an IDE that includes different tools to support model-driven development. The repository is passive. Each component has control to access the repository. Some of the examples of repository approach are: university management system, library information system, database management systems (DBMS), computer-aided software engineering tools (CASE), etc.

3) Client-server architecture

The Client-Server pattern is a distributed architecture where the functions in the system are performed using clients and servers. The client is a service requester. The client requests a specific service to serve using the internet. The server responds. Though the model is distributed, the independent services can be implemented on a single computer. The most common example of client-server software architecture is email, the world wide web, etc. Again, an important benefit is a separation and independence. Services and servers can be changed without affecting other parts of the system. This pattern is described in Table 8.3.

Name	Client-server
Description	The description is information of a set of services and servers. Clients are users of these services and access servers to make use of them.
Example	A network with a client-server system
When used	During shared database and functions. As servers can be replicated, it is also used when the load on the system is dynamic.
Advantages	The client need not have all the functions; servers can be distributed across the network.
Disadvantages	There may be a denial of service to clients due to security attacks or server failures. The performance may not be good as the functions are dependent on servers across the network.

Table 8.3: Client-Server Architecture

Figure 8.19 is an example of a system that is based on the client-server model. The client programs can use the services related to printing, database, email, and other services through the internet.

The most important advantage of the client-server model is that it is a distributed architecture. The systems can be networked with many distributed processors. Any server can be added or upgraded without affecting the rest of the system.

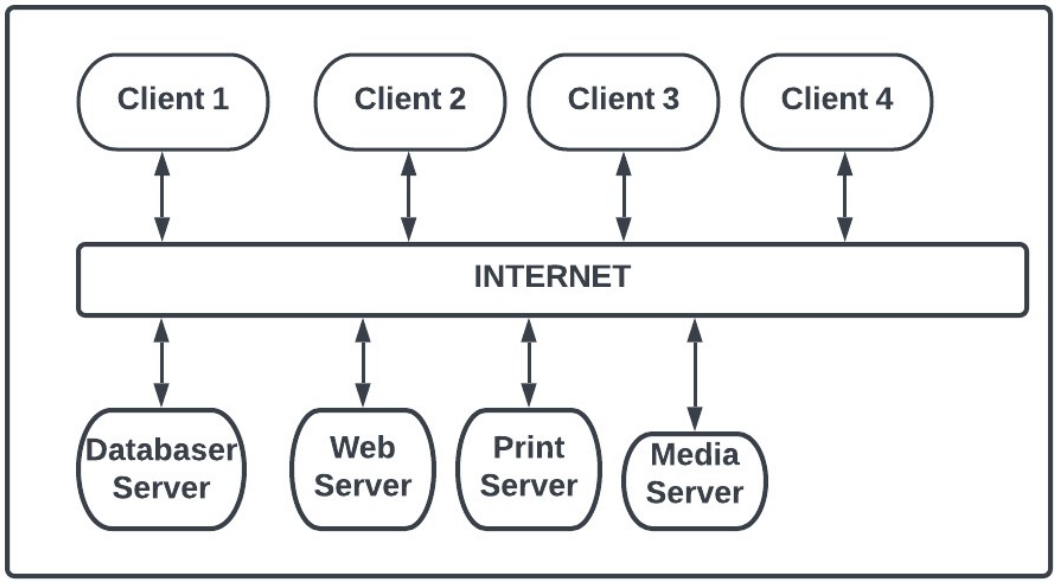

Fig 8.19: A client-server architecture for a computer network

4) Pipe and Filter architecture

The pipe and filter architecture is inspired by the Unix operating system where the different processes were linked using pipes. This model shows the execution or runtime organization of a system. Here the filter is a component that transforms the data and processes it as input. The pipe connects different stream of data that is flowing in a sequence. Each processing step is implemented as a transform. Input data flows through these transforms until converted to output. The transformations may execute sequentially or in parallel. The data can be processed by each transform item by item or in a single batch. This pattern is described in Table 8.4.

Name	Pipe and filter

Description	Filters are discrete and perform data transformation. The data flow in a pipe from one component to another for processing.
Example	A pipe and filter system is used in compiler construction where input is one language and output is in another, Wikipedia where one link of information connects to another.
When used	Commonly used in data-processing applications where inputs are processed in separate stages to generate related outputs. The data processing may be batch processed or transaction-based.
Advantages	Many businesses processes match the pipe and filter approach, which makes it suitable and easy to implement. Easy to understand and supports transformation reuse. It can be used in both sequential or concurrent.
Disadvantages	The order of data transfer should be decided in advance. If there are many independent components, then there will be a lot of system overhead on data transformation. If there are incompatible data structures, then it is difficult to use one output and input in another component.

Table 8.4: The Pipe and Filter pattern

An example of this type of system architecture, used in a compilation of a C program is shown in Figure 8.20.

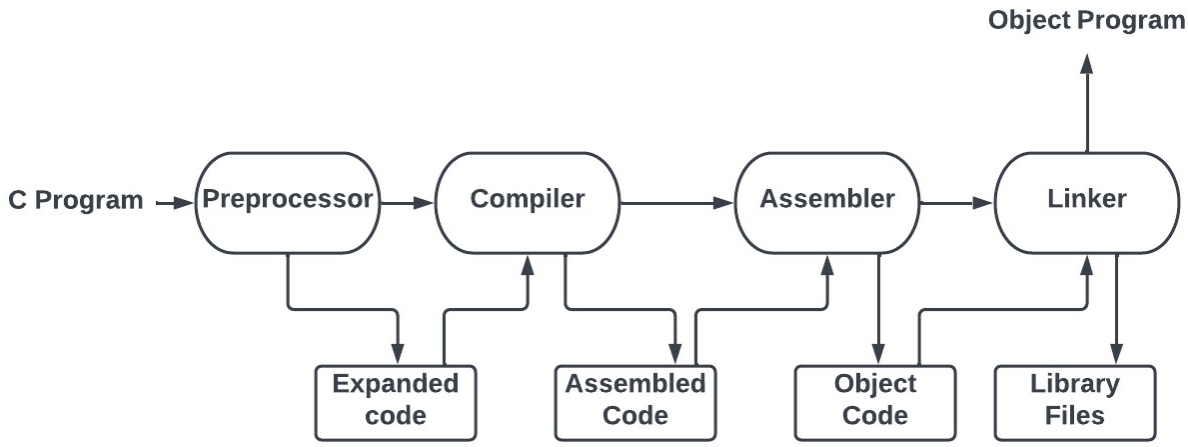

Fig 8.20: Pipe and Filter model in the compilation of C program

The first component is the pre-processor where the code is expanded for resolving constants and macros and then passed to the compiler. The compiler transforms the pre-processed code into assembly code. The assembler generates an object file. The linker links the object file to different library files used in a program. The output of the linker is an executable file.

Pipe and filter systems are also used in batch processing systems and embedded systems where there is limited user interaction. An invoice processing of customers to confirm the payment, issuing receipts, sending reminders, etc. is a good example of a pipe and filter model. If the system is interactive that includes events such as mouse clicks or menu selection, then processing the data becomes difficult in a sequential manner. The pipe and filter model is not suitable in such applications as the sequential stream cannot be confirmed.

8.9 Multiple Choice Questions (MCQs)

1. In OOD, data and functions are combined using:
   a) Encapsulation
   b) Polymorphism
   c) Inheritance
   d) Aggregation
2. One class can derive several subclasses in OOD, this is given as:

a) Polymorphism

b) Inheritance

c) Message Passing

d) Abstraction

3. The different functions carried in several objects in a time sequence is shown by:

a) Use case diagram

b) Sequence diagram

c) Class diagram

d) Object diagram

4. The interaction between the actors and the system but no functions are shown in a diagram in OOD is:

a) Use-case diagram

b) Data flow diagram

c) State machine diagram

d) Sequence diagram

5. The requirements between the external user and the system can be gathered using a UML diagram:

a) Component diagram

b) Use case diagram

c) Sequence diagram

d) State diagram

6. The static data structures in software are denoted by:

a) Object diagram

b) Class diagram

c) Deployment diagram

d) Use-case diagram

7. The class and object diagram is classified into the categories:

a) Structural UML

b) Behavioral UML

c) Interaction UM:

d) Flowchart

8. Which is of the following is not a UML diagram

a) State diagram

b) Component diagram

c) Flow diagram

d) Object diagram

9. Object in OOD is:

a) An instance of a class

b) Member of a class

c) Interface

d) Not instance of a class

10. Which view is used to show the system hardware and how software components are distributed across the processors in the system:

a) Process

b) Physical

c) Logical

d) Data

8.10 Review Questions

1. Describe the use case diagram and class diagram with a suitable example.
2. Explain the relationships used in OOD with a suitable example.
3. What are use cases and actors? Explain the use case diagram for any system.

4. Identify the objects if you have been assigned to design a bank automation software. Define the appropriate entities, data members' operations, interfaces, and relationships required between different objects.
5. You have been asked to prepare and deliver a presentation to a non-technical manager to justify the hiring of a system architect for a new project. Write a list of bullet points setting out the key points in your presentation in which you explain the importance of software architecture.
6. A software system will be built to allow drones to autonomously herd cattle on farms. These drones can be remotely controlled by human operators. Explain how multiple architectural patterns can fit together to help build this kind of system.
7. Suggest an architecture for a system that is used to sell and distribute music on the Internet. What Architectural patterns are the basis for your proposed architecture?
8. With a suitable diagram, explain the layered architecture design model.
9. Explain client-server architecture with a suitable diagram.
10. Briefly define the top-down and bottom-up design models.
11. Compare top-down and top-down design approaches.

Answers to MCQ

1: a	2: b	3: b	4: a	5: b	6: b	7: a	8: c	9: a	10: b