UNIT- I

1 Concepts of Software Modeling

Syllabus

Software Modeling : *Introduction to Software Modeling, Advantages of modeling, Principles of modeling.* ***Evolution of Software Modeling and Design Methods :*** *Object oriented analysis and design methods, Concurrent, Distributed Design Methods and Real-Time Design Methods, Model Driven Architecture (MDA), 4+1 Architecture, Introduction to UML, UML building Blocks, COMET Use Case–Based Software Life Cycle.* ***Requirement Study :*** *Requirement Analysis, SRS design, Requirements Modeling.* ***Use Case :*** *Actor and Use case identification, Use case relationship (Include, Extend, Use case Generalization, Actor Generalization), Use case template.*

Contents

Part I : Software Modeling

1.1 Introduction to Software Modeling

- Building a software is similar to build a house. One should gather all the necessary needs and wants before constructing the house. The blueprint of house must be prepared before the actual construction work. Just similar to that before creating any software it is necessary to collect all possible requirements. The model for the software must be prepared on the paper.
- The **basic issue** in development of quality architecture is creating right software and in figuring out how to write less software.
- There are many elements that contribute to a successful software organization : One such common element is the use of modeling.
- Modeling is a well accepted engineering technique for designing the software. The architectural or mathematical models can be build for creating the software.
- Modeling is basically required in order to better understand the system and to communicate the purpose of the software to others.
- **Definition of model :** The model is simplification of reality.
- A model provides the blueprint of the system. A good model includes those elements that have broad effect and omit those minor elements that are not relevant to given level of abstraction.
- Every system may be described from different aspects using different models. A model may be **structural** - emphasizing the organization of the system. A model can be **behavioral** - emphasizing the dynamics of the system.
- **Purpose of modeling :** We model the system because we want to understand the system which we are building.

1.2 Advantages of Modeling

There are four advantages of the modeling and those are -

1. Models help to **visualize the system.**
2. Models allow us **to specify the structure** or behaviour of the system.
3. Using models we get the template using which the **system can be constructed.**
4. Models help in **documenting the decisions** that are made during the development of the software.

Thus we build the models of complex systems because we can not comprehend such systems in its entirety.

Every project gets benefited from the modeling. If the project is more complex then there are lots of chances of failure. The most useful systems normally becomes complex after some time. Hence if we do not model the system at its early stage then it becomes difficult to build the model for such system as it evolves.

1.3 Principles of Modeling

Following are the **four basic principles** of modeling -

1. **The choice of which entity to model has great effect on how to solve the problem and how a solution must be shaped.**
- If the system is a database system then the focus might be on entity - relationship models. The behaviors will be transferred to triggers and stored procedures.
- If the structured approach is to be adopted then the models will be algorithmic - centric with data flowing from one process to another.
- If the object oriented approach is adopted then architecture will center around classes and patterns that interact with each other.
- Any of these approach can be adopted for a given application, but among all these approached the object oriented approach is superior.

2. **Every model can be expressed with different levels of precision.**
- There is variety of stakeholders for the system. These include users, developers, analyst and so on. The user and analyst will focus on what is there in the system, the developer will focus on how is done in the system. Thus both of these stakeholders will examine the system from different levels of details at different times. In any case the best model is that model that allows to view it with required details.

3. **The best models are connected to reality.**
- It is best to have models that have a clear connection to reality.
- Such type of models can help in identifying the ideal conditions or to find out the fatal characteristics.

4. **There can not be single model or view to represent the system.**
- If you want to build a house no single blueprint is sufficient. You might need floor plan, elevations, electricity plans, plumbing plans and so on. In the same manner, multiple views are required to capture the breadth of the system.
- The variety of models created (using different views) can be built and studied separately but still must be interrelated.

- In object oriented modeling the system can be modeled using several views such as use case view, design view, interaction view, implementation view and deployment view.
- Depending upon the type of the system to be developed some views may be more important than others. For instance : In GUI based system static and dynamic use case views are more important or in web based applications the deployment models are more useful.

Part II : Evolution of Software Modeling and Design Methods

1.4 Evolution of Software Modeling

- In 1960s before developing the programs, graphical notations such as **flowcharts** were often used as a documentation tool or as a design tool. Program can be divided into modules. Each module is normally developed by separate person and implemented as **subroutine** or **function.**
- With the growth of structured programming in early seventies, two approaches came up - **top down design** and **stepwise refinement.** These approaches are designed with the intent of systematic approach for structured program design.
- The **Data Flow Diagram(DFD)** and **data structured design** are the two approaches are developed in mid to late 1970s. These are **structured design** approaches.
- In data structured design, the emphasis is on first designing data structures and then designing the program structures based in the data structures.
- In database designing, the Entity Relational modelling (ER) is used as a logical design method.
- Major contribution for design of concurrent and real time systems came in the late 1970s. with the introduction of MASCOT notation. Based on data flow approach, MASCOT formalized the way tasks communicate with each other either through channels for message communication or through pools.
- In 1980s, the work on applying structured analysis and structured design to concurrent and real time systems led to the development of **Real Time Structured Analysis and Design(RTSAD)** and the **Design Approach for Real-Time Systems(DARTS)** methods.
- In early 1980s the Jackson System Development (JSD) was introduced. From JSD's point of view, the design is simulation of real world and emphasis was on modelling the entities in the problem domain by using concurrent tasks.

1.5 Object Oriented Analysis and Design Methods

- In the mid to late 1980s, the popularity of object oriented programming led to design of object oriented analysis and design methods. The emphasis of these methods was on modelling the problem domain, information hiding and inheritance.
- Information hiding is a design approach which tells the software designers to design the **self - contained software modules.** That means, the modules should be designed in such a way that even if one module is changed, there must be little or no impact on other modules.
- Booch introduced object oriented concepts into design initially with information hiding in **object based design** of Ada based systems. This later extended to **object oriented design.**
- **Object oriented analysis** methods apply object - oriented concepts to the analysis phase of software life cycle. In this analysis technique, the **real - world objects** are identified from the problem domain and they are then **mapped** with **software objects.**
- During object oriented analysis, initially static analysis is carried out to build the **static object modelling.** The class models and object models are created during static object modelling.
- The object modelling is basically done by two approaches - **UML** and **OMT**. Prior to UML the most widely used notation was OMT.
- The Object Modelling Technique **(OMT)** suggested by Rambaugh emphasised on **dynamic modelling.** The dynamic modelling is carried out by constructing **state chart diagram** and **sequence diagrams** to show the sequence of interactions among the objects.
- In 1992, Jackson introduced the **use case concept** to represent the functional requirements. Jackson also used the sequence diagram in which the interacting objects are the objects that participate in use case diagrams.
- The **use case** concept has profound impact on **modern** object oriented software development.
- The Unified Modeling Language **(UML)** notation was originally developed by Booch, Jacobson, and Rambaugh (1997) to integrate **use case modelling, static modelling** and **dynamic modelling.**

1.6 Concurrent, Distributed Design Methods and Real - Time Design Methods

Following is a survey of concurrent, distributed and real - time design methods -

1) The **Concurrent Design Approach for Real-Time Systems (CODARTS)** method is designed for concurrent and real - time system design. The emphasis of this method is on information hiding and concurrent task structuring.

2) **Octopus** is a real time design method based on use cases, static modelling, object interactions and statecharts.

3) **ROOM** is an object oriented real - time design method that uses CASE(Computer Aided Software Engineering) tool called **ObjecTime.**

4) Bhur introduced **use case map** which is based on use case modelling. It addresses the issue of dynamic modelling of large scale systems.

5) Earlier version of **COMET method** is also used for designing concurrent, real - time and distributed applications.

1.7 Model Driven Architecture (MDA)

- Model Driven Architecture (MDA) is an approach of producing code from abstract, human elaborated modelling diagrams.
- The Model Driven Architecture is launched by Object Management Group (OMG) in 2001.
- This approach is based on forward engineering, which means that the code is produced from human elaborated diagrams or models.
- There are three models in MDA -
  (1) The **CIM** stands for **Computation - Independent Model** which combines the requirements and domain models. It models the system in terms of how it will interact with its environment.
  (2) The **PIM** stands for Platform Independent Model. It is a **design model.** It describes the internal structure of model without the hosting platform.
  (3) A **PSM** stands for Platform Specific model. It is the **implementation model.** It adds concepts from the hosting platform to the hosting platform. Examples of platforms include .Net, J2SE, J2ME, J2EE, CORBA.

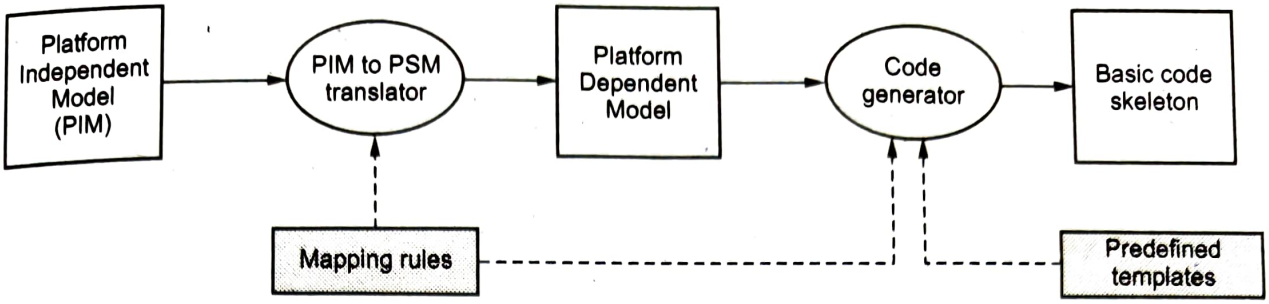


Fig. 1.7.1 Model Driven Architecture(MDA)

- The abstract Computer Independent Model (CIM) works as basis for Platform Independent Model (PIM), which is then transformed into Platform Specific Model (PSM), PSM then transform into code. Code in MDA is considered as a concrete model.
- CIM is model captures system requirements and vocabulary of problem domain, that is independent of computers. CIM creation is optional. PIM expresses business semantics independent to any platform. PIM updated with platform specifications to generate platform specific model. Source code is generated for the targeted platform from PSM.

1.8 The 4+1 Architecture

SPPU : Dec.-16,19, April-17,19, May-17, March-20, Marks 5

- The architecture of the system can be described by 4 + 1 types of views Design, interaction, implementation and use case view.

Design view :

1. This view supports the functional requirements of the system.
2. It represents the services that are provided to the end user by the system.
3. With UML the static aspects of the system are represented by the class diagram and the object diagram, whereas the dynamic aspects of the system are represented by the interaction diagram, state diagram and the activity diagram.
4. The internal structure diagram of the class is focus in this view.

Interaction view :

1. This view is intended to represent flow of control among the various parts of the system.
2. This view focuses on the performance, scalability, throughput of the system.

Implementation view :

1. This view contains a collection of artifacts which are combined and assembled together to form the physical system.
2. Various files are linked together to produce implementation of the system.

Use case view :

1. This view supports all the activities that are required to deploy all the hardware technology required to execute the system in the targeted environment.
2. The focus of this view is on distribution, delivery and installation of various parts in order to make up the whole system.
   - Following Fig. 1.8.1 represents such type of software architecture.

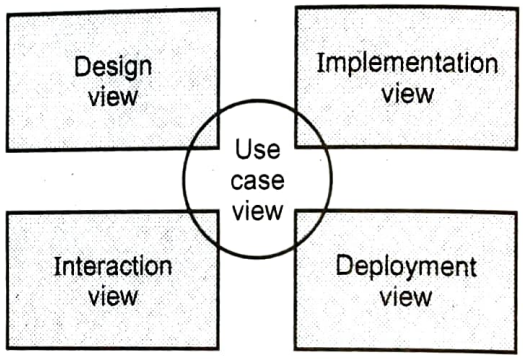


Fig. 1.8.1

Review Questions

1. *Explain different views of software architecture ? What is 4+1 architecture view model ? Explain with suitable diagram ?* **SPPU : Dec.-16, End Sem, Marks 5**
2. *Explain 4-1 architecture view.* **SPPU : April-17, In Sem, May-17, End Sem, Marks 5**
3. *What is 4+1 view architecture? Why it is called as 4+1 view architecture ?* **SPPU : April-19, In Sem, Marks 5**
4. *Explain 4+1 view architecture of UML.* **SPPU : Dec.-19, End Sem, March-20, In Sem, Marks 5**

1.9 Introduction to UML

Definition : The Unified Modeling Language (UML) is a standard diagramming notation for specifying, visualizing, constructing and documenting the artifacts of software systems, as well as for business modeling and other non-software systems.

Goals of UML

The goals of UML are -

1. UML is intended to provide ready- to use and expressive visual modeling language in order to develop the effective system model.

2. The modeling must be independent of the programming language or platform but at the same time the automatic code that can be generated by this tool must support the modern programming languages.

3. This tool supports the high level concepts such as collaborations, framework, patterns and components.

4. It helps to integrate the organizational standards in the software systems.

1.9.1 Basic Building Blocks

- The basic building blocks include -
  1. Things 2. Relationships 3. Diagrams
- **Things** are the basic and abstract entities in model. **Relationships** tie these things together. The **diagrams** is the collection of all these things along with relationships.
- There are four kinds of things -

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

1.9.1.1 Things

1. **Structural things :** These are the **static part of a model,** representing elements that are either conceptual or physical. The structural things are also called as **classifiers.**

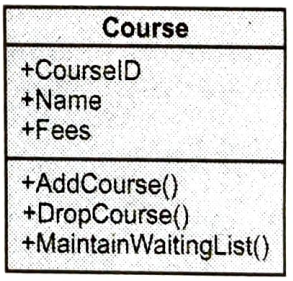
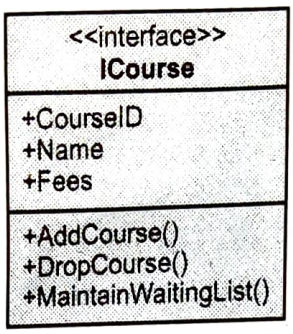


Fig. 1.9.1 Course

- A class is description of set of objects. It contains set of attributes, operations, relationships and semantics. A class implements one or more interfaces. Graphically it can be represented using the rectangle. It is normally divided in three sections. The first section contains the name of the class, the second section contains the list of attributes and the third section contains the corresponding functionalities.
- **Interface** is a collection operations that specify the service of the class or component. The interface describes the externally visible behaviour of a class.
- Interface may represent external behavior of the system. It might represent the complete behavior or part of that behavior. While declaring the interface the keyword <<interface>> is used above the name. A small circle might be attached to the class box line.



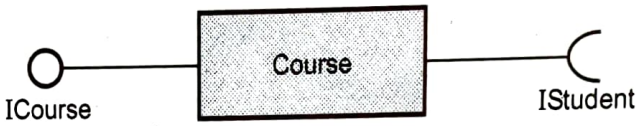


Fig. 1.9.2 Interface

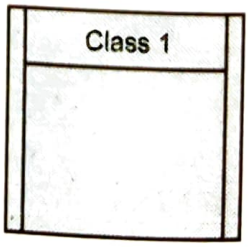
- **Collaboration** represents the interaction with other element which provides the cooperative behavior. A given class or object can take part in several collaborations, this can be represented by collaborations. The collaboration can be represented by dotted elliptical shape. Fig. 1.9.3 represents the collaboration -

Fig. 1.9.3 Collaboration

- **Use Case** represents by specific scenario by means of set of actions that a system can perform. These actions are driven by the actor. Use case is used to represent the behavioural things of the system. Use case is realized by collaboration. Use case is represented by a solid elliptical shape. For example - Refer Fig. 1.9.4.

Fig. 1.9.4 Use case

- **An active class** indicates that, when instantiated, the class controls its own execution. Rather than being invoked or activated by other objects, it can operate standalone and define its own thread of behavior. In other words this class initiates a thread or a process. For example - CommandProcessor is an active class when user submits some command then based on that command particular process is invoked for execution. The active class is represented by double lines on left and right sides.



Example

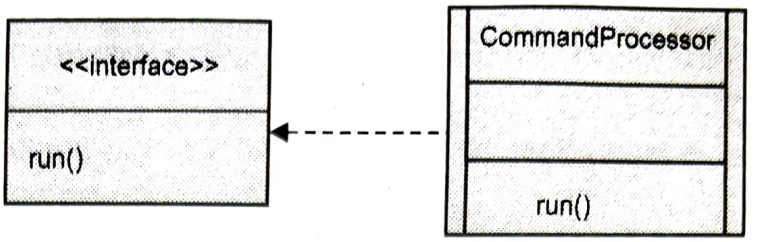


Fig. 1.9.5

- **Component :** The component is a structured class representing modular part of the system with encapsulated contents.

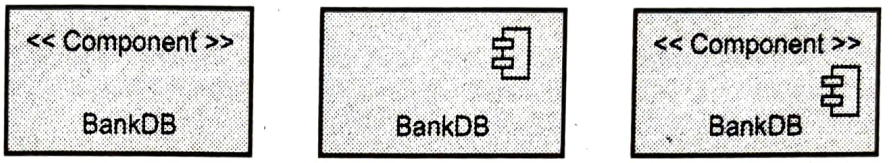
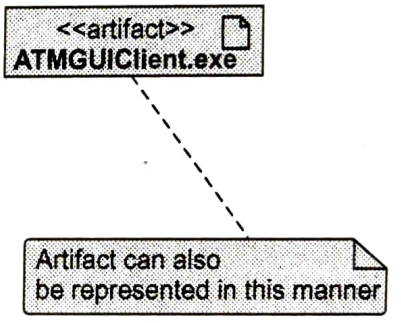


Fig. 1.9.6 Three ways of representing component

A stereotype is mentioned in the gulliments. A component represented in rectangle without using stereotype is treated as class.

- An **artifact** i.e. a concrete physical element such as .exe, .jar or .war file can be represented as artifact. Sometimes the client scripting files such as XML or HTML may also represent artifact.
- There are two types of nodes used in deployment diagram - (Refer Fig. 1.9.7)



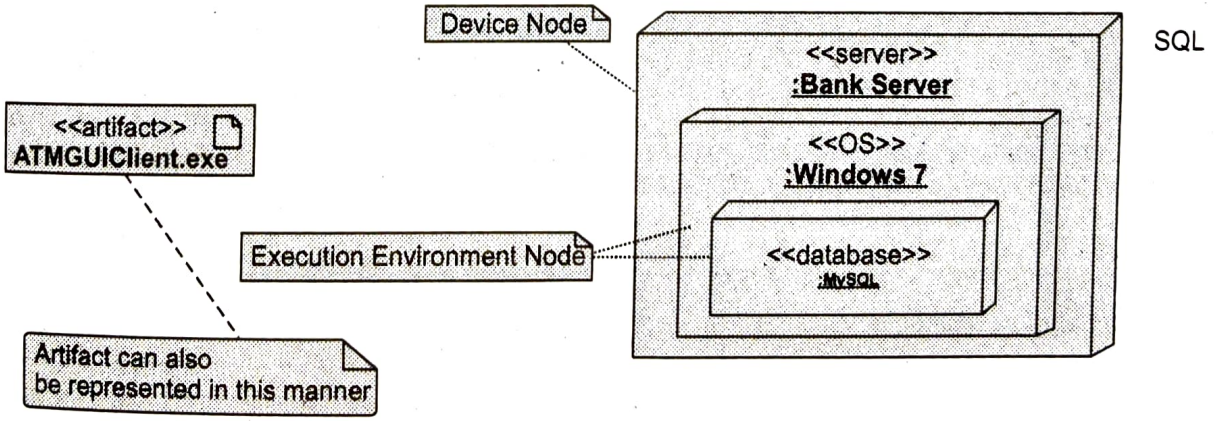


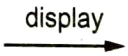
Fig. 1.9.7 Deployment diagram for ATM system

- **Device Node** - It represents the physical computing device having processor and memory. This node is responsible for executing software system. For example Bank Server, Computer, Mobile Phone and so on.
- **Execution Environment Node (EEN)** - This is a software computing resource which typically executes on device node. Following are some choices of EEN -
  - Database Engine which requests the structured queries and executes them.
  - Web browser which executes client side scripting such as HTML, DHTML, JavaScript, applets and so on.
  - Workflow engine
  - Servlet container.
  - Operating system software

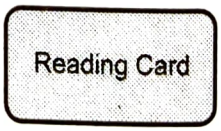
- Thus the elements such as classes, interfaces, collaborations, use cases, active classes, artifacts and nodes are the basic structural components of the UML model.

2. **Behavioral things :** These are **dynamic part of the model.** These are verbs of the model representing behavior of the system. There are three types of behavioral things -

   i. Interaction represents the set of messages exchanged among the set of objects within particular context to accomplish certain task. It is represented by a direct line as shown in figure



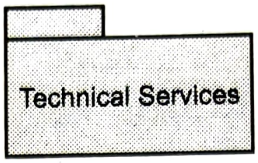
   ii. **State machine** represents the **events** and **states** of object and behavior of the object in reaction to event. The state machine shows the life cycle of the object. Sometimes it is not possible to show each and every event in the state diagram. Graphically state is represented by a rectangle with rounded corners.



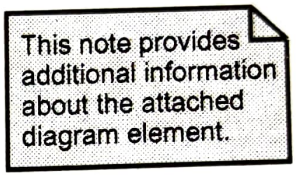
   iii. **Activity** is the behavior that specifies the steps performed by the process. The focus of this model is on set of objects that interact with each other. A step of activity is called action. Graphically activity is represented by a rectangle with rounded corners.

3. **Grouping things :** are the kind of organization of the UML models. The primary kind of grouping is called **package.**

- Package is a general purpose mechanism for organizing the design itself. Package is purely conceptual. Package can group anything for instance classes, other packages, use cases and so on.
- A package can be represented as follows -



4. **Annotational things :** These are for explanation pupose. The most commonly used annotational thing is **note.** It can be used for specifying comments or constraints. Graphically it can be represented as a rectangle with a **dog-eared corner.**

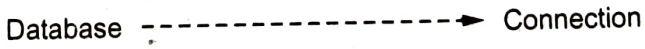


1.9.1.2 Relationship in UML

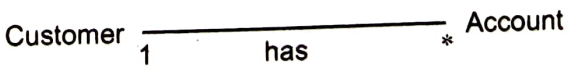
- There are four types of relationships in relationships in UML -

1. Dependency	2. Association
3. Generalization	4. Realization

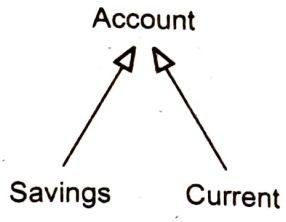
- **Dependency :** This kind of relationship exists when one element is dependant another element. This is semantic kind of relation. When two elements are represented by this relationship, changes made in one element can affect the another element. This relation can be represented by a dashed line possibly directed and having label.



- **Association :** This is the simple relationship to link two objects. **Association** is a special kind of relationship. It is a **whole part relationship.** Graphically it is represented by a solid line possibly directed and must be labeled. For example



- **Generalization :** This relationship shows the parent child relationship. Child shares the behavior of the parent. Graphically it is represented by a solid line with a arrow head at one end. The arrow head is hollow pointing to the parent. For example



- **Realization :** It is a relationship between two model elements, in which one model element (the client) realizes the behavior that the other model element (the supplier) specifies. Several clients can realize the behavior of a single supplier. Realization shows the relationship between an Interface an the class that provides the implementation for the interface. It is represented by a dashed line with an hollow arrow head.

1.9.1.3 Diagrams in UML

- Diagrams are the graphical representation of the set of the elements. The diagrams can be drawn to represent the system from different perspectives. The same elements may appear in different diagrams. The diagram contains the collection of elements and relationship among these elements. Various types of diagrams that can be drawn in UML are -
  1. Class diagram
  2. Object diagram
  3. Component diagram
  4. Use case diagram
  5. Sequence diagram
  6. State diagram
  7. Activity diagram
  8. Deployment diagram
  9. Package diagram
  10. Timing diagram
- **Class diagram :** It contains the set of classes, interfaces, collaborations and their relationships. The class diagram represents the static design view of the system. These diagrams are commonly drawn in UML.
- **Object diagram :** These diagrams show the relationship among the different objects. Objects are the instances of the classes, hence the object diagrams represent real or prototypical cases. These diagrams represent the static design view of the system.

- **Component diagram :** This diagram contains the collection of components and connectors. It represents the static design view of the system. It shows the encapsulated classes, interfaces, ports and internal structure of the system.
- **Use case diagram :** These diagrams are drawn to represent the static design view of the system. The use cases represent the **behaviour** of the system. It consists of collection of use cases, actors(special type of classes) and their relationships.
- **Sequence diagram :** These are kind of interaction diagrams consisting of set of objects, roles along the messages. They are specially useful for modeling the behaviour of the system.
- **State diagram :** The state diagrams represent the states, transitions, activities and events. The event -ordered behaviour of the object is emphasized.
- **Activity diagram :** This diagram represents the flow of control or data step by step occurring in computations. These diagrams emphasize on representation of functionality. These diagrams represent the dynamic view of the system.
- **Deployment diagram :** It addresses the static deployment view of the architecture. The run time processing of the components is represented by these diagrams.
- **Package diagram :** These diagrams show the organizational units and their dependencies.

1.10 COMET

SPPU : April-16,18,19, Marks 5

- The Collaborative Object Modeling and Design Method or COMET uses the UML notation to describe the design.
- This is an iterative, use case driven and object oriented software development method.
- This design method makes use of various features such as information hiding, classes, inheritance, and concurrent tasks.

Phases of COMET

1. **Requirements modeling :** In this phase a requirement model is developed in which the functional requirements of the system are described in terms of actors and use cases.
2. **Analysis modeling :** The static and dynamic models are built during this phase. The static model is represented by means of class diagram and dynamic model is represented using state chart diagram.

3. **Design modeling :** During this phase the software architecture of the system is designed. In this phase the analysis model is mapped to an operational environment.

4. **Incremental software construction :** During this phase, a subset of a system is selected to be constructed for each increment. This subset is determined by choosing the use case and participating objects to be included in this increment. In this phase the software is gradually constructed.

5. **Incremental software integration :** During this phase, the integration testing of each software increment is performed. The integration test cases are developed for each use case.

6. **System testing :** In this phase, the complete system is tested against its functional requirements.

Review Questions

1. *Explain COMET and phases of COMET.* **SPPU : April-16, In Sem, Marks 5**
2. *Comment on how COMET is useful in software design and analysis.* **SPPU : April-18, In Sem, Marks 5**
3. *What is COMET ? Explain phases of COMET.* **SPPU : April-19, In Sem, Marks 5**

1.11 Use Case - Based Software Life Cycle

In the **Software Development Life Cycle(SDLC)**, there are various phases such as project planning, Analysis, Design, Implementation and support or maintenance.

Out of these phases we will discuss the design phase. Following Fig. 1.11.1 represents various activities that can be performed during the design phase of SDLC. (See Fig. 1.11.1 on next page.)

Various design activities are explained as follows -

1. Design and integrate network : Sometimes the system has to be designed along with the new network. If this is the case then the new network needs to be designed. Various technical issues may rise while making the new system to work on the network such as reliability, security, synchronization and so on. However, many times instead of creating new network, the existing networking system is preferred.

2. Design application architecture software : In this activity the decision about the **structure and configuration of new system** and design of actual computer software is taken. All the components of the system depends upon each other, and desired **configuration of application architecture** help in taking the important design decisions.

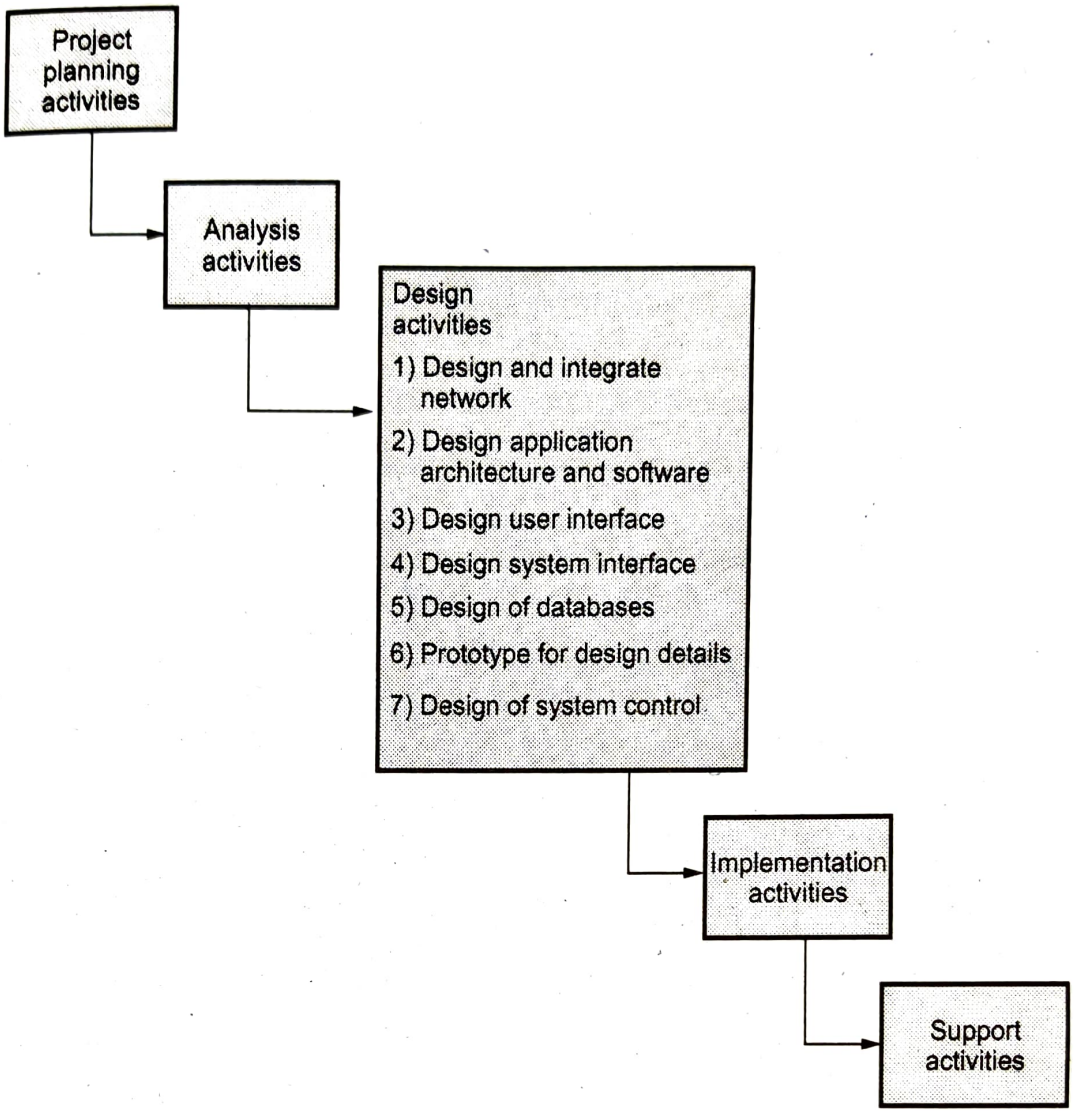


Fig. 1.11.1 Design activities

In this designing the **application architecture** detail how all system activities will actually be carried out is specified.

3. Design user interface : This activity defined how user will interact with the system. The design of various components such as windows, dialog box, mouse interaction, icons, sound and video commands is specified. Sometimes User interface designers assist in this activity. **Interface designers** are specialist in user interface design.

4. Design system interface : The design of system interface defines how one system component interchanges information with other system component.

5. Design of databases : During the design and integration of databases, the performance of database system must be considered. The performance tuning helps to work system efficiently.

6. Prototype for design details : Prototype is a sample working model used for testing some software component before committing major resources to particular configuration. Many times rapid application prototype (RAD) is prepared to test the desired software component.

7. Design of system control : The design of system control involves designing of system controls for all the other activities such as user interface, system interface, application architecture, database, network design.

These system design activities are carried out and then the set of design documents can be prepared. These design documents must be consistent and integrated to provide comprehensive set of specifications for a complete system.

Part III : Requirement Study

1.12 Requirement Analysis

Requirement engineering is the process of

- *Establishing the services that the customer requires from a system.*
- *And the constraints under which it operates and is developed.*

The requirements themselves are the descriptions of the system services and constraints that are generated during the requirements engineering process.

What is a requirement ?

A requirement can range from a high-level abstract statement of a service or of a system constraint to a detailed mathematical functional specification.

The requirement must be open to interpretation and it must be defined in detail.

Types of requirements

The requirements can be classified as

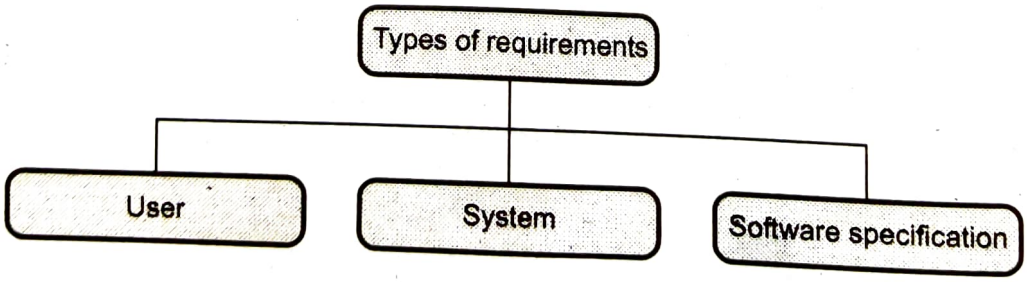


Fig. 1.12.1 Types of requirements

- **User requirements**

It is a collection of statements in natural language plus description of the services the system provides and its operational constraints. It is written for customers.

- **System requirements**

It is a structured document that gives the detailed description of the system services. It is written as a contract between client and contractor.

- **Software specification**

It is a detailed software description that can serve as a basis for design or implementation. Typically it is written for software developers.

1.12.1 Need for Requirements to be Stable and Correct

Following are those reasons which specify that there is a need for the requirements to be stable and correct -

1) As requirements are identified and analysis model is created the software team and other stakeholders **negotiate the priority**, availability and relative cost of the requirement. During this negotiation there are chances that the requirements may get changed.

2) During requirement analysis, each requirement is validated against the needs of the customer.

3) If the requirements are stable, correct and unambiguous then they remain unchanged throughout the requirement analysis process. And finally the working model that satisfied customers need can be created effectively.

1.13 Requirements Engineering

SPPU : Dec.-15, May-16, Marks 5

- Requirement engineering is the process **characterized** for achieving following **goals** -
  - Understanding customer requirements and their needs
  - Analyzing the feasibility of the requirement
  - Negotiating the reasonable solutions
  - Specification of an unambiguous solution.
  - Managing all the requirements of the project
  - Finally transforming the requirements into the operational systems

- Requirement engineering process performs following **seven distinct functions**

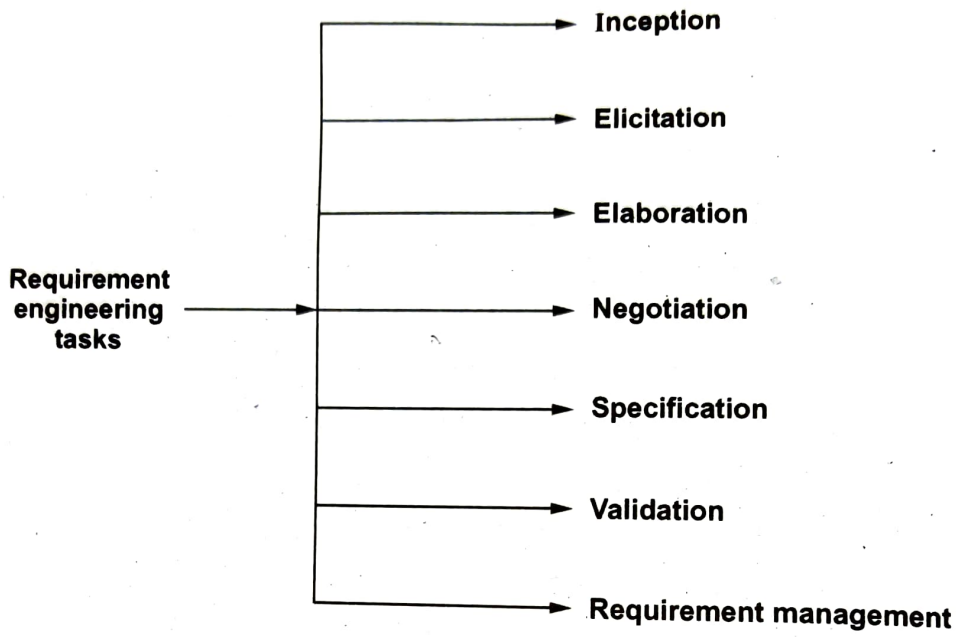


Fig. 1.13.1 Requirement engineering tasks

Let us now discuss these tasks in detail -

1.13.1 Inception

The inception means specifying the **beginning** of the software project. Most of software projects get started due to business requirements. There may be poter demand from the market for a particular product and then the specific software pro needs to be developed.

There exist several **stakeholders** who define the **business ideas.** Stakeholders m an **entity** that takes active participation in project development. In software pro development, the stakeholders that are responsible for defining the ideas are busi managers, marketing people, product managers and so on. Their role is to do ro feasibility study and to identify the scope of the project.

During the **inception** a set of **context free questions** is discussed. The purpos inception is to -

1. Establish the basic understanding of the project.
2. Find out all possible solutions and to identify the nature of the solution.
3. Establish an effective communication between developer and the customer.

1.13.2 Elicitation

Before the requirements can be analyzed and modelled they must undergo through the process of elicitation process. Requirements elicitation means **requirements discovery.** Requirements elicitation is very difficult task.

Following are the reasons for : ***why it is difficult to understand customer wants ? -***

1. Customer sometimes is unable to specify the **scope of the project**. Sometimes customers specify too many technical details and this may increase the confusion.
2. There is difficulty in **understanding the problem**. Sometimes customer could not decide what are their needs and wants. Sometimes they have got poor understanding of capabilities and limitations the existing computing environment.

   Sometimes customers find it **difficult to communicate** with the system engineer about their needs. Sometimes customers may have got **some conflicting requirements.** This ultimately results in specifying **ambiguous requirements.**
3. As project progresses the needs or requirements of the customers changes.. This creates a problem of **volatility**.

To overcome these problems the requirements gathering must be done very systematically.

1.13.3 Elaboration

- Elaboration is an activity in which the information about the requirements is **expanded** and **refined**. This information is gained during inception and elicitation.
- The **goal** of elaboration activity is to prepare a technical model of software functions, features and constraints.
- The elaboration consists of several **modelling and refinement tasks**. In this process several **user scenarios** are created and refined. Basically these scenarios describe how end-user will interact with the system.
- During elaboration, each user scenario is parsed and various **classes** are identified. These classes are nothing but the business entities that are visible to end user. Then the **attributes** and **services** (functions) of these classes are defined. Then the relationship among these classes is identified. Thus various UML (Unified Modelling Diagrams) are developed during this task.
- Finally the **analysis model** gets developed during the elaboration phase.

1.13.4 Negotiation

Sometimes customer may **demand for more** than that is achieved or there are certain situations in which customer demands for something which cannot be achieved in **limited business resources**. To handle such situations requirement engineers must convince the customers or end users by solving various **conflicts**. For that purpose, requirement engineers must ask the customers and stakeholders to **rank** their **requirements** and then priority of these requirements is decided. Using **iterative approach** some requirements are eliminated, combined or modified. This process continues until the users' satisfaction is achieved.

1.13.5 Specification

- A specification can be a **written document, mathematical** or **graphical model, collection of use case scenarios** or may be the **prototypes.**
- There is a need to develop a **standard specification** in which requirements are presented in consistent and understandable manner.
- For a large system it is always better to develop the specification using natural language and in a written document form. The use of graphical models is more useful for specifying the requirements.
- Specification is the final work product of requirement engineering process. It describes the functions, constraints and performance of computer based systems.

1.13.6 Validation

- Requirement Validation is an activity in which requirement specification is analyzed in order to ensure that the requirements are specified unambiguously. If any inconsistencies, omissions and errors are identified then those are corrected or modified during the validation.
- The most commonly used requirement validation mechanism is **Formal Technical Review (FTR)**. In FTR, the review team validates the software requirements. The review team consists of requirement engineers, customers, end users, marketing person and so on. This review team basically identifies conflicting requirements, inconstancies or unrealistic requirements.

1.13.7 Requirement Management

Definition : Requirements management is the process of managing changing requirements during the requirements engineering process and system development.

Why requirements get change ?

- Requirements are always **incomplete** and **inconsistent**. New requirements occur during the process as business needs change and a better understanding of the system is developed.
- System customers may specify the requirements from business perspective that can conflict with end user requirements.
- During the development of the system, its business and the technical environment may get changed.

Requirement management process

Following things should be planned during requirement process.

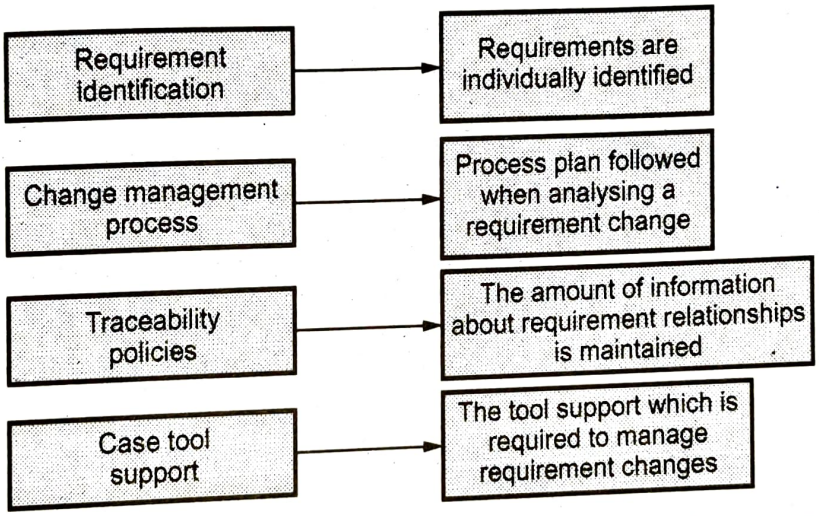


Fig. 1.13.2 Planning in requirement management process

- **Traceability** is concerned with relationship between **requirements, their sources** and the **system design**.

Various types of traceability are

1. **Source traceability** - These are basically the links from requirement to stakeholders who propose these requirements.
2. **Requirements traceability** - These are the links between dependant requirements.
3. **Design traceability** - These are the links from requirements to design.

- Case tool support is required for

1. Requirement storage
2. Change management
3. Traceability management

- Traceability information is typically represented by a data structure **Traceability matrix.** If one requirement is dependent upon the other requirement then in that row-column cell 'D' is mentioned and if there is a weak relationship between the requirements then corresponding entry can be denoted by 'R'.

For example

Requirement ID	A	B	C	D	E	F
A		D			R	
B			D			
C				R		
D			D			R
E						
F	R			D		

For mentioning the traceability of small systems usually the traceability matrix is maintained.

Review Questions

1. *Why it is difficult to gain a close understanding of what the customer wants ?*

SPPU : Dec.-15, End Sem, Marks 5

2. *Explain the tasks included in requirement engineering activities.*

SPPU : May-16, End Sem, Marks 5

1.14 Requirement Modeling

SPPU : Dec.-14, Marks 5

Analysis modelling approach	
Structured approach	**Object oriented approach**
The analysis is made on data and processes in which data is transformed as separate entities.	The analysis is made on the classes and interaction among them in order to meet the customer requirements.
Data objects are modelled in such a way that data attributes and their relationship is defined in structured approach.	Unified Modelling Language(UML) and unified processes are used in object oriented modelling approach.

But the commonly used analysis model combines features of both these approaches because the best suitable analysis model bridges the software requirements and software design.

Following are the elements of analysis model -

- Scenario based elements
- Flow-oriented elements
- Behavioural elements
- Class based elements

Following Fig. 1.14.1 illustrates the elements of analysis model.

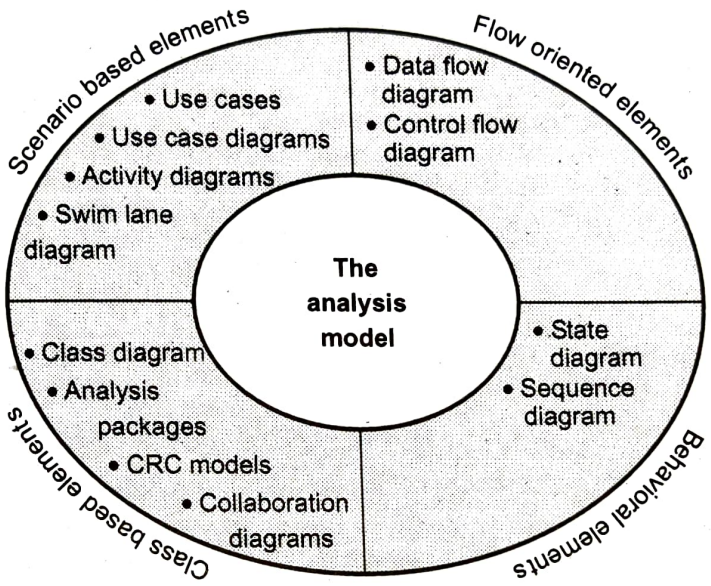


Fig. 1.14.1 Analysis model

Review Question

1. *What are different approaches or elements of a requirement analysis model ?*

SPPU : Dec.-14, End Sem, Marks 5

1.15 SRS Design

SPPU : Dec.-14, Marks 5

- The software requirements provide a basis for creating the Software Requirements Specifications (SRS).
- The SRS is useful in estimating **cost, planning team activities, performing tasks** and **tracking** the team's progress throughout the development activity.
- Typically software designers use IEEE STD 830-1998 as the basis for the entire Software Specifications. The standard template for writing SRS is as given below.

Document Title

Author(s)
Affiliation
Address
Date
Document Version

1. Introduction

1.1 Purpose of this document

Describes the purpose of the document.

1.2 Scope of this document

Describes the scope of this requirements definition effort. This section also de any constraints that were placed upon the requirements elicitation process, such schedules, costs.

1.3 Overview

Provides a brief overview of the product defined as a result of the requireme elicitation process.

2. General description

- Describes the general functionality of the product such as similar syst information, user characteristics, user objective, general constraints placed design team.
- Describes the features of the user community, including their expected exper with software systems and the application domain.

3. Functional requirements

This section lists the functional requirements in ranked order. A functio requirement describes the possible effects of a software system, in other words, *what* system must accomplish. Each functional requirement should be specified in follow manner

- Short, imperative sentence stating highest ranked functional requirement.

1. Description

A full description of the requirement.

2. Criticality

Describes how essential this requirement is to the overall system.

3. **Technical issues**
   Describes any design or implementation issues involved in satisfying this requirement.

4. **Cost and schedule**
   Describes the relative or absolute costs of the system.

5. **Risks**
   Describes the circumstances under which this requirement might not able to be satisfied.

6. **Dependencies with other requirements**
   Describes interactions with other requirements.

7. **Any other appropriate**

4. Interface requirements

This section describes how the software interfaces with other software products or users for input or output. Examples of such interfaces include library routines, token streams, shared memory, data streams and so forth.

4.1 User Interfaces

Describes how this product interfaces with the user.

4.1.1 GUI

Describes the graphical user interface if present. This section should include a set of screen dumps to illustrate user interface features.

4.1.2 CLI

Describes the command-line interface if present. For each command, a description of all arguments and example values and invocations should be provided.

4.1.3 API

Describes the application programming interface, if present.

4.2 Hardware Interfaces

Describes interfaces to hardware devices.

4.3 Communications Interfaces

Describes network interfaces.

4.4 Software Interfaces

Describes any remaining software interfaces not included above.

5. Performance requirements

Specifies speed and memory requirements.

6. Design constraints

Specifies any constraints for the design team such as software or hardware limitations.

7. Other non functional attributes

Specifies any other particular non functional attributes required by the system. Such as :

7.1 Security

7.2 Binary Compatibility

7.3 Reliability

7.4 Maintainability

7.5 Portability

7.6 Extensibility

7.7 Reusability

7.8 Application Compatibility

7.9 Resource Utilization

7.10 Serviceability

... others as appropriate

8. Operational scenarios

This section should describe a set of scenarios that illustrate, from the user's perspective, what will be experienced when utilizing the system under various situations.

9. Preliminary schedule

This section provides an initial version of the project plan, including the major tasks to be accomplished, their interdependencies, and their tentative start/stop dates.

10. Preliminary budget

This section provides an initial budget for the project.

11. Appendices

11.1 Definitions, Acronyms, Abbreviations :

Provides definitions terms, and acronyms, can be provided.

11.2 References

Provides complete citations to all documents and meetings referenced.

1.15.1 Characteristics of Good SRS

Following are some characteristics of a good SRS -

1. Correct

- The SRS must be correct.
- That means all the requirements must be correctly mentioned, or the requirements must be realistic by nature.
- For instance : While developing a word processing software, if there is a requirement for spell check facility and if software cannot find the spelling errors from the document, then that means requirement is incorrect.

2. Complete

- To make the SRS complete, it should be specified what a software designer wants to create a software.
- The SRS is **said to be complete** only in following situations -
  1) When SRS consists of all the requirements related to functionality, performance, attributes, design constraints or external interfaces.
  2) When labels and corresponding references are mentioned for all the figures, diagrams and tables in the SRS.
  3) When expected responses to the input data is mentioned by considering validity and invalidity of an input.

3. Unambiguous

- When requirements are understood correctly then only unambiguous SRS can be written.
- Unambiguous specification means only one interpretation can be made from the specified requirements. In other words, there should be an **unique interpretation** of each statement in SRS.
- If for particular term there are multiple meanings then, those terms should be mentioned in glossary with proper meaning. The requirements should not be mentioned in the same manner.

- After preparing SRS, it should be **reviewed** by third party.
- Normally SRS is written in some **natural language** like SRS. The SRS can b examined with the help of language processors, so that lexical, syntactic an semantic errors can be exposed.

4. Consistent

If there are not **conflicts** in the specified requirements then SRS is said to b consistent. Three types of conflicts that may occur in a SRS -

i) **Logical or temporal conflict :** When one requirement specifies that event X shoul occur before event Y and if another requirement specifies that event Y shoul occur before event X.

ii) **Characteristics conflicts of real-world object :** If one requirement suggests to mak use of "radio button" and other specifies the "Check box button", then it represent conflicting characteristics.

iii) **Two different descriptions about the same real world object :** If one requiremen is specifying "Enter" and other specifies "submit" then it describes one and th same action.

5. Stability

In SRS, it is not possible to specify all the requirements. The SRS must contain all th essential requirements. Each requirement must be clear and explicit.

6. Verifiable

- The SRS should be written in such a manner that the requirements that ar specified within it must be satisfied by the software.
- For instance - "The GUI should look good". This requirement is not verifiabl because one cannot specifically define "what is mean by good ?".

7. Traceable

- If origin of requirement is properly given or references of the requirements ar correctly mentioned then such a requirement is called as **traceable requirement.**
- Various types of traceability are -

i) **Forward Traceability :** Each requirement is referred in the SRS document by it unique name or reference number.

ii) **Backward Traceability :** If the reference to the requirement is mentioned in earlie document, then it is backward traceability.

Review Question

1. *Describe IEEE template for eliciting software requirement specifications. What information is produced as a consequence of the requirement gathering ?*

SPPU : Dec.-14, End Sem, Marks 5

Part IV : Use Case

1.16 Introduction to Use Case

SPPU : Aug.-15,17, April-18, Dec.-16, May-17, March-20, Marks 5

- Use cases are scenarios for **understanding the requirements.**
- The use case model is used in project development, planning and documentation of the system.
- **Definition :** Use case is a sequence of transactions in a system whose task is to yield results of measurable value to an individual actor of the system.
- Graphically use case is represented as ellipse. For example

Basic Terminologies : Following components are used in use case modeling.

(1) **Use Cases :** Use case is a special flow of events through the system. Various events can be grouped together and call each group a use case class. By grouping the use cases, the complexity of the design can be made manageable.

(2) **Actors :** An actor is a user playing a role with respect to system. Actor is a key factor for correctly finding the use cases.

(3) **In a System :** It means that actors communicate with the system's use case.

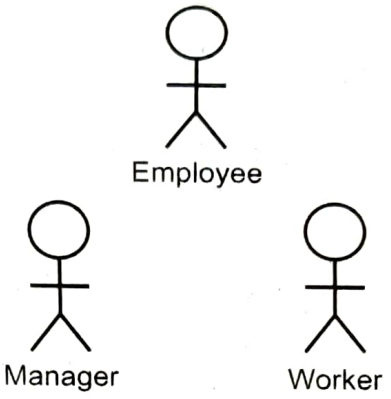
(4) **Measurable Value :** Use case can help actor to perform a task that has some identifiable value. For example - borrowing a book is something of value for a member of the library.

(5) **Transaction :** The transaction is an atomic set of activities that are performed fully or not at all.

1.16.1 Actor Identification

- Actor represents a role when users interact with the use cases.
- The actor can be a human, device or other system. For example - in a Company Information System, an Employee is an actor.

- Actors can be connected to the use cases only by association. This associati represents that actor and corresponding use cases communicate with each other sending and receiving messages.
- For example -



Guideline for actor identification

The actors can be identified by answering following questions -

(1) Who is using the system ? or who is affected by the system ?

(2) Who affects the system ? Which user group is needed by the system to perform functions.

(3) Which external hardware or other system is required by this system to perfo task.

(4) What kind of problems are solved by this system ?

(5) How do users use this system ?

Two three rule

- The two-three rule is for identifying actors. The rule is stated as "start wi naming at least two, preferable three, people who could serve as the actors in t system. Other actors can be identified in the subsequent iterations.
- For example - While designing the use case model for hospital managemen system, the two important actors that come in picture are patient and doctor, th iteratively various actors such as Nurses, Laboratory Assistants, Ward-bo Receptionist, Accountant and so on.

1.16.2 Actor Classification

Actor is something that interacts with the System under Discussion (SuD). There a three kinds of actors -

1. **Primary actor :** The primary actors are the actors that interact with the system order to achieve the user goals. For example, customer is the primary actor for t use case On-line purchase system.

2. **Secondary actor :** The secondary actors are used in conjunction with the primary actors in order to support for the main services. The secondary actors support the system in such a way that the primary actors can perform their task. For example payment validation system is a supporting actor for the on-line purchase system.

For example -

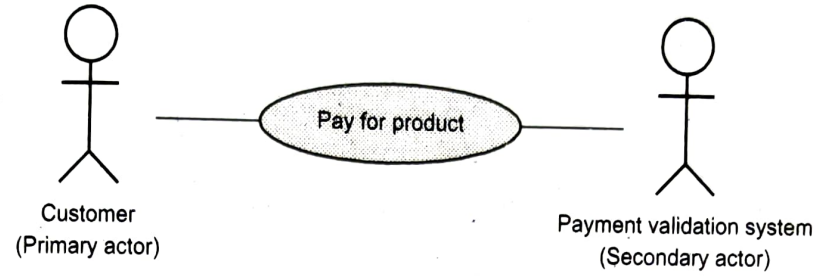


Fig. 1.16.1

1.16.3 Actor Generalization

- In the context of use case modeling the **actor generalization** refers to the relationship which can exist between two actors and which shows that one actor (descendant) inherits the role and properties of another actor (ancestor).
- The generalization relationship also implies that the **descendant actor** can use all the use cases that have been defined for its ancestor.
- For example -

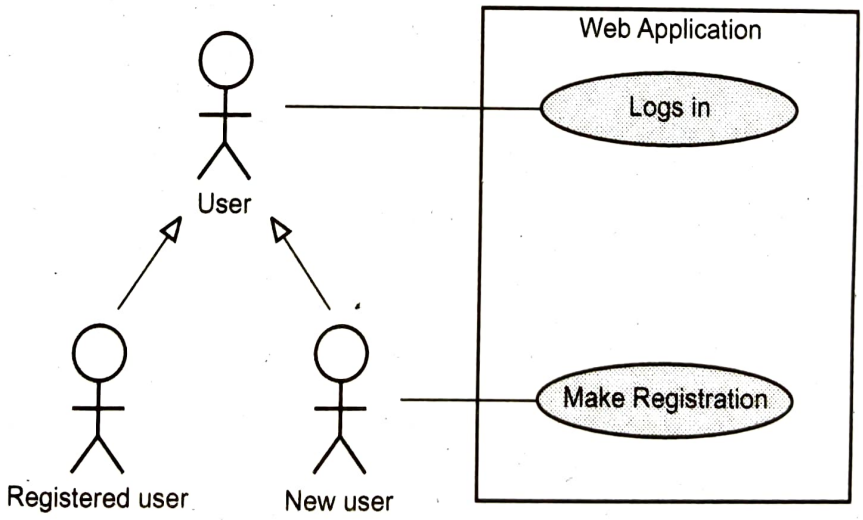


Fig. 1.16.2 Actor generalization

1.16.4 Use Case Identification

The **next important step** immediately after identifying the actors is - to describe the way by which these **actors interact** with the system. This should be done iteratively.

Following are steps used for **identification of use cases -**

(1) For each actor, find the **function** that the actor should perform or the function that can be performed by the system. For example - Customer Pays for the product.

(2) **Name** the use cases. For example - "Pay for Product"

(3) **Describe** use cases - For example i) makes Cash Payment ii)Makes Credit Card Payment iii) Makes cheque Payment

1.16.5 Communication

- The use cases specify what is there in the system and the flow of events specify how things work in a system.
- When the flow of events is to be described then include following things -
  - How and when use case starts and ends.
  - Interaction of various actors with use case.
  - The objects that are exchanged.
  - Basic and alternate flow of behavior.
- Example - In context of course reservation system, the use case for the activity of reserving a course is as given below -

Main scenario :

1. Student selects for reservation
2. Student views the available courses and selects the desired course.
3. Student fills up the form.
4. Student submits the form.
5. Student receives an eligibility message and proceeds for payment.
6. Student selects the mode of payment.
7. Student pays the fees.
8. Student confirms the reservation.

Exceptional flow of events

The list of course is unavailable.

The desired course is not available.

Exceptional flow of events

The student is not eligible for desired course.

Exceptional flow of events

For online payment, the balance is insufficient.

1.16.6 Use / Include and Extend Association

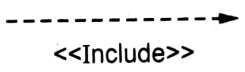
Several use cases can be related with each other. Similarly the use cases are also related to the actors who are intended to achieve the goal of the system. The use cases are related to any other entity by using include, extend and generalization relationship.

1 Include

- This is the most commonly used relationship which denotes that a given use case may include another.

The use case at the arrow head position is the use case which is included by use case on the other side of the arrow. When there are multiple steps to carry out the single task then this task is divided into the sub-functions and these sub-functions are denoted by the use cases. For example -

It is denoted by



Use Case Template	
Use Case	On - line shopping
Primary Actor	Customer
Goal in Context	To monitor all the functionalities required to establish the session
Scenario	1. Customer logs in to purchase the product online 2. Customer browses through the products. 3. Customer selects the product and adds it to the shopping cart. 4. The contents of the shopping cart are displayed. 5. The product and shipping information is confirmed by the customer. 6. The customer pays for the product
Extensions	6a The customer pays by credit card (include relationship) 6b The customer pays by the cheque (include relationship) 6c The customer pays by cash (include relationship)

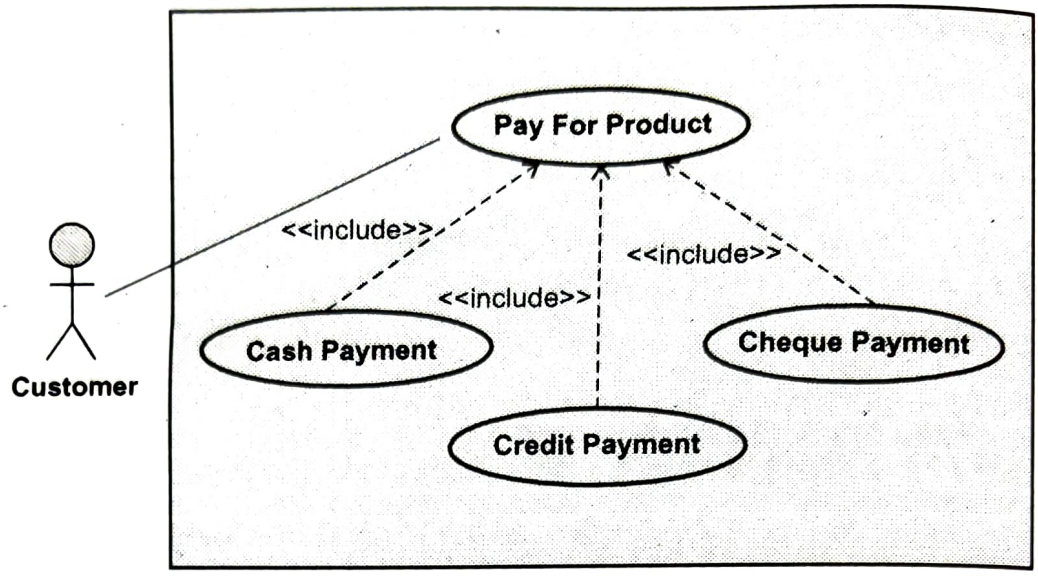
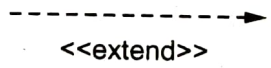


Fig. 1.16.3 Use case for payment for the product (Include- relationship)

2. Extend

- The extend relationship is used when an extended use case is connected to base use case. While using the extend use the extension points can be explic shown in the base use cases. Following are the reasons why extensions occur -
  1. When some part of the use case is optional and we need to show sepa behavior of the system.
  2. When we want to show a subflow of the system when some specific conditi occur.

This relationship is denoted by -



- The extension is conditional. That means its execution is dependent on what happened while executing the base use case. These conditions are described by extend relationship.
- The extension use case may access and modify attributes of the base use case

Example : **Online purchase system,** when user browses for the product, it m browse either by the product name or by product ID.

Use Case template	
Use Case	On-line shopping
Primary Actor	Customer

Goal in Context	To monitor all the functionalities required to establish the session
Scenario	1. Customer logs in to purchase the product online 2. Customer browses through the products. 3. ... 4. ... 5. The customer pays for the product
Triggers : Extensions points	Customer browses the catalog. 2a Customer browses the product by entering the product name 2b Customer browses the product by entering the product ID.

Extension Point

The extension points are used when we use the extend relationship. The extension points are the labels in the base use case which are referred by the extending use case. For example

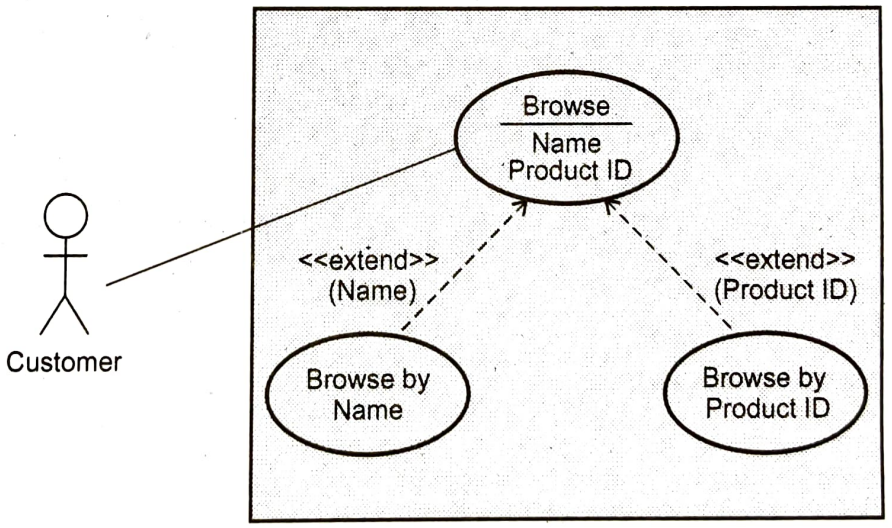


Fig. 1.16.4 Use case for browsing product (Extend - relationship)

3. Generalization

- In this type of relationship there are two types of use cases - the parent use case and the child use cases.
- A parent use case may be specialized into one or more child use cases that represent more specific forms of the parent. The child use cases inherit all the structure, behavior and relationships of parents.

- Generalization is used when we find two or more use cases having commo structure and behavior and purpose.

The test word used for generalization relationship is **"kind of"**. It is denoted by

For example : For the **Purchase system** the customer can place an order via telepho or using internet.

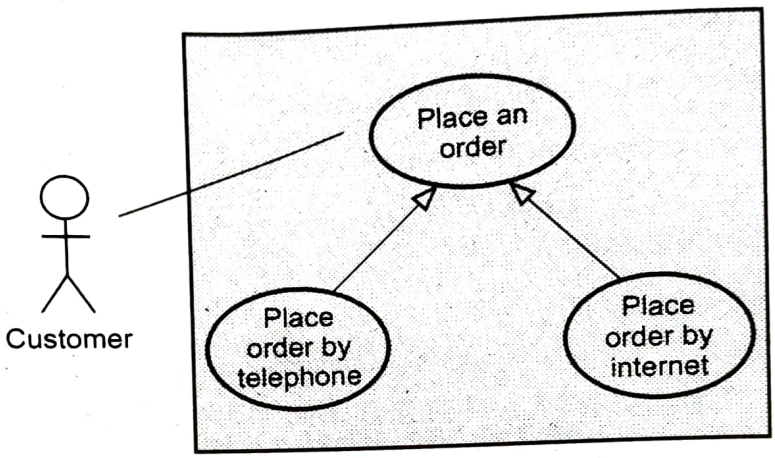


Fig. 1.16.5 Generalization Use case

1.16.7 Use Case Realization

- Graphically use case is represented as ellipse. For example -

Use cases specify the behavior of the system. However, it does not specify how implement this behavior.

For implementing the behavior of the use case, collection of classes(static structure) and other elements have to work together. Thus for implementation purpose, the collection of elements that belong to static and dynamic structure is formed. This collection is called as collaboration.

The realization if use case is possible with the collaboration using the realization relationship.

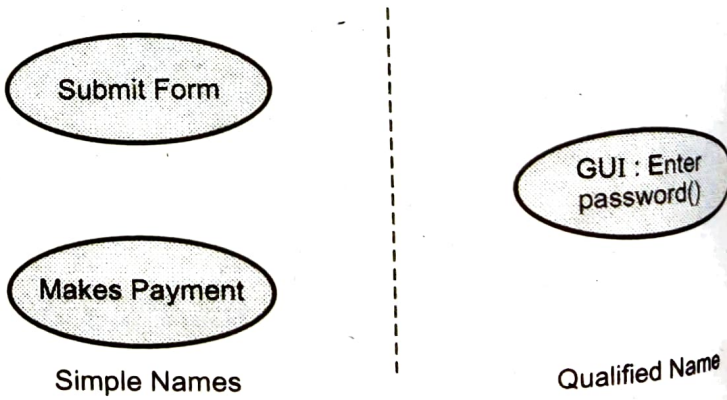


Fig. 1.16.6 Names to use cases

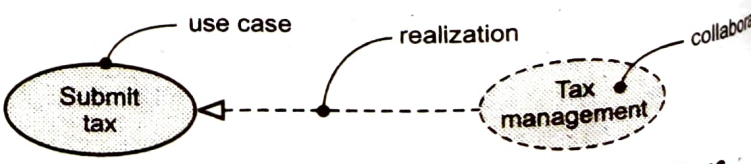


Fig. 1.16.6 (a) Use case and collaborations

1.16.8 Use Case Diagram

Definition : Use case is a collection of use cases and actors and the relationships among them.

- Use case diagram contains following things -
  - Subject
  - Use cases
  - Actors
  - Various relationships such as generalization, include and exclude.
- Use case diagram also contains the notes and constraints to add more semantic to the modeling.
- For grouping various elements into a group use case diagram may also contain packages.
- Sometimes the instances of use cases can be placed in the model for understanding execution of the system.

1.16.9 Examples on Use Case Diagram

Example 1.16.1 *Draw the use case diagram for demonstrating various relationships in it.*

Solution : See Fig. 1.16.7.

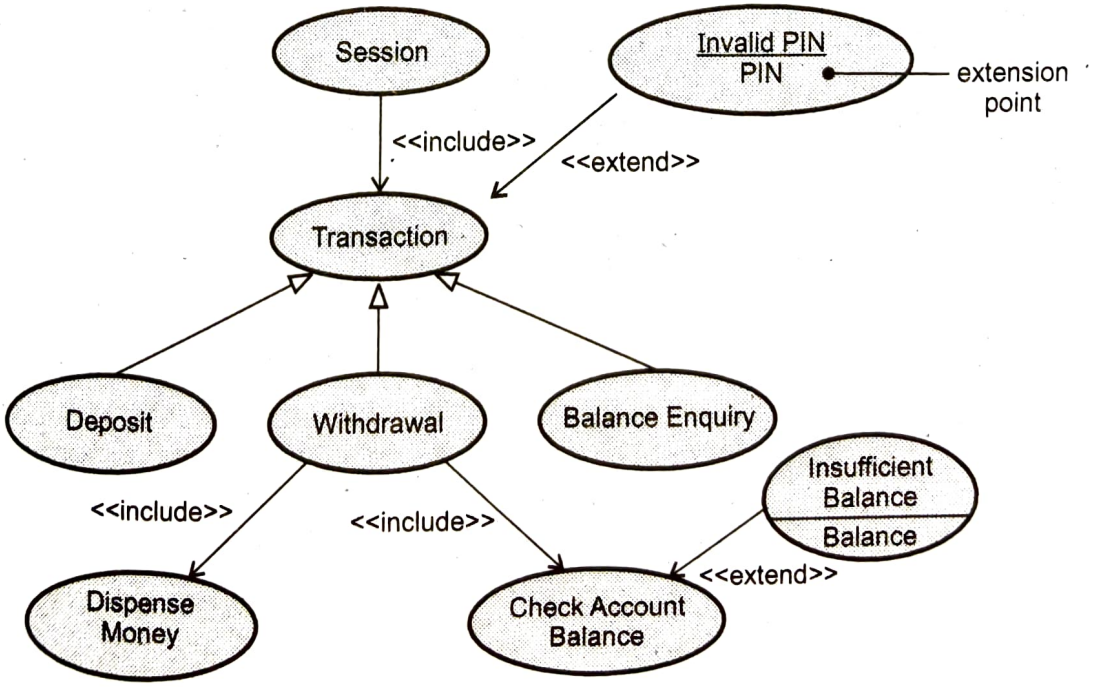


Fig. 1.16.7 Withdrawal of money from ATM

Example 1.16.2 *Describe use case "Validate User" in modeling an ATM system.*

Solution : There are two actors for the Bank ATM system. The **Customer** and the **Bank**. There are two types of customers - current account holder and savings account holder.

Following are the steps by which the customer interacts with the ATM system -

1. Customer inserts the card.
2. The ATM system validates the card.
3. If there is valid card entered by the customer then the ATM system requests the customer for entering PIN, The customer enters the PIN.
4. The PIN entered by the customer is validated.
5. If the valid PIN is entered then only the customer is prompted for performing the transactions.
6. Customer can select the desired transaction such as Withdrawal of money, Deposition of fund or simply enquires for the available balance amount.

The use case for Validate User is as given below -

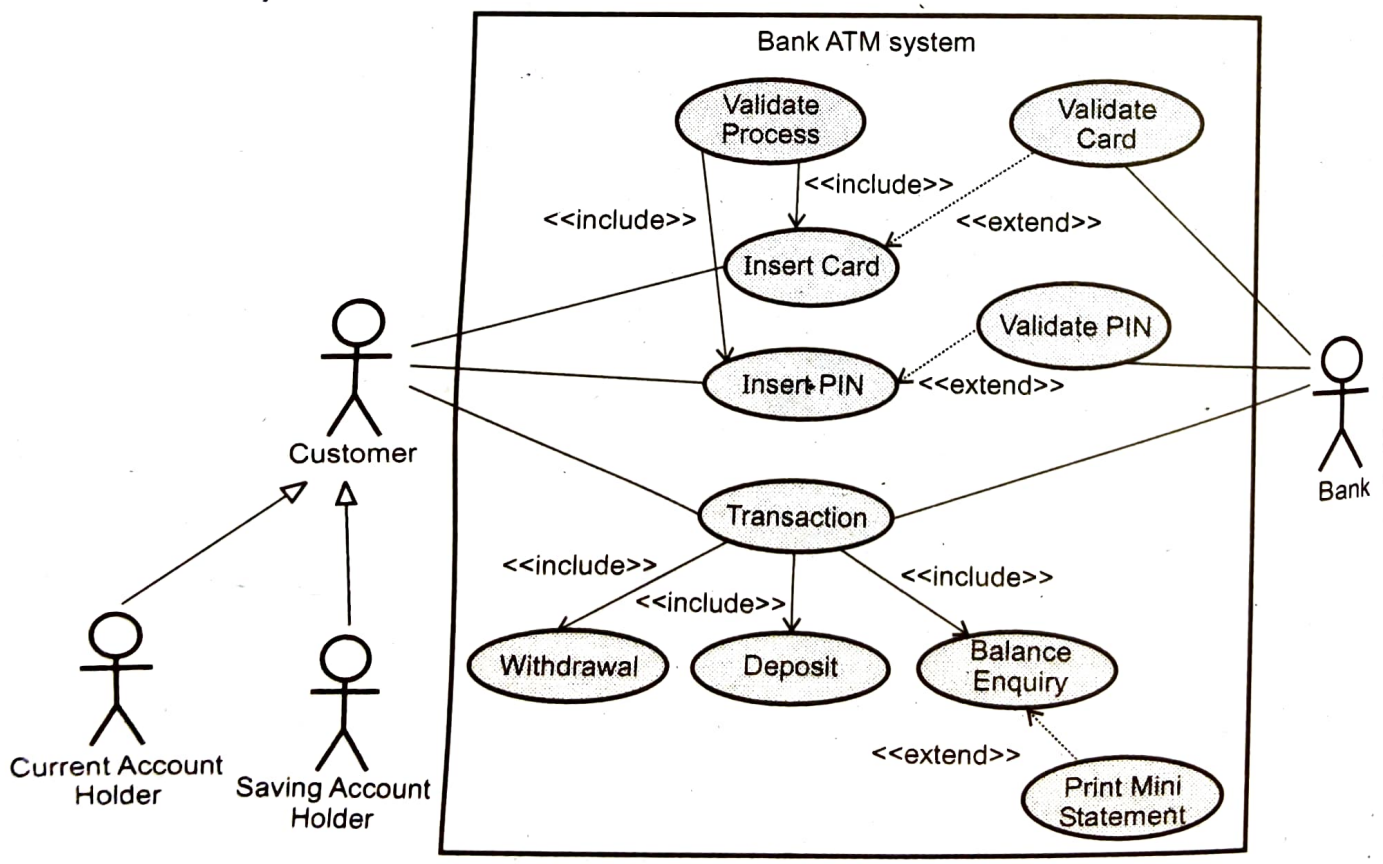


Fig. 1.16.8 Validate User use case

Example 1.16.3 *Consider a software system like 'examination system'. Assume that there are use cases defined like 'view marks', 'input exam id','view practical marks', optionally send/forward my mark sheet as an Email'. Show how use case relationships like includes, generalizations and extends can be used to appropriately model above use cases and their relationships in context of use case diagram.*

Solution :

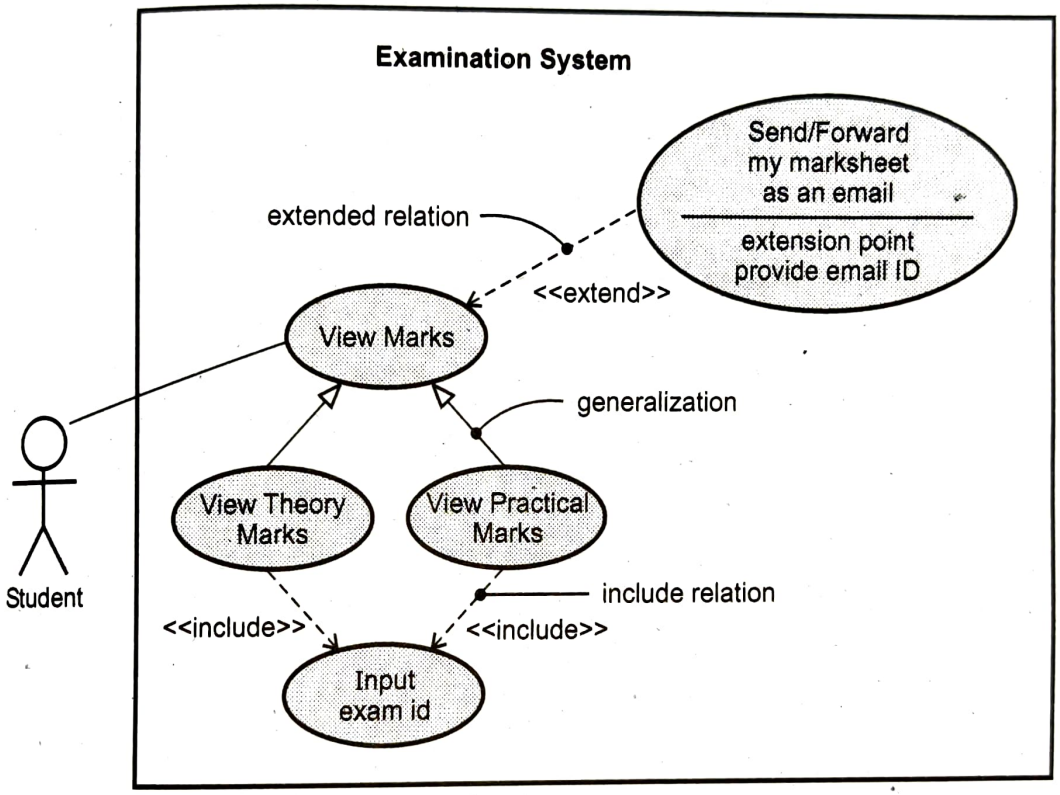


Fig. 1.16.9 Use case diagram for examination system

Example 1.16.4 *Give use case diagram for hospital management system.*

Solution : (See Fig. 1.16.10 on next page.)

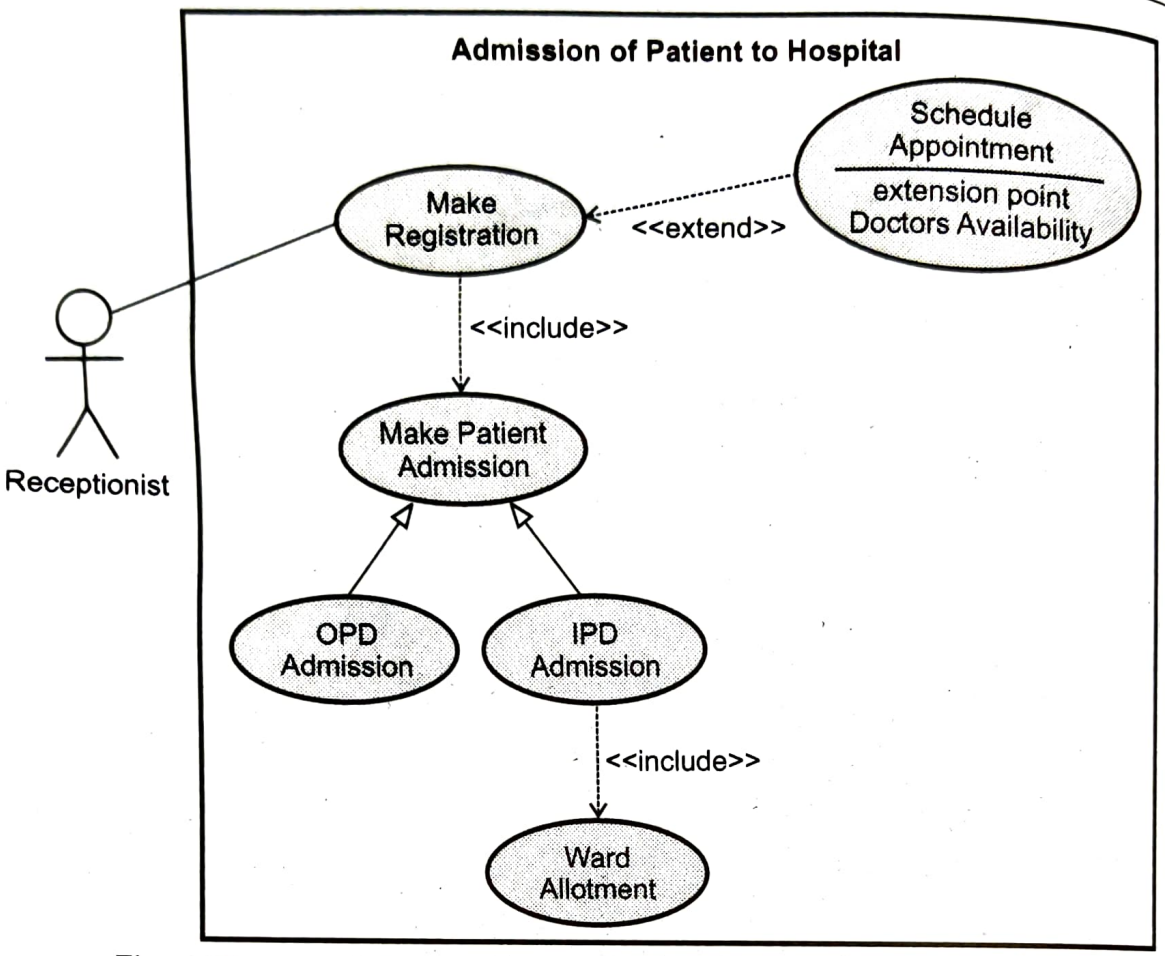


Fig. 1.16.10 (a) Use case diagram for hospital management system

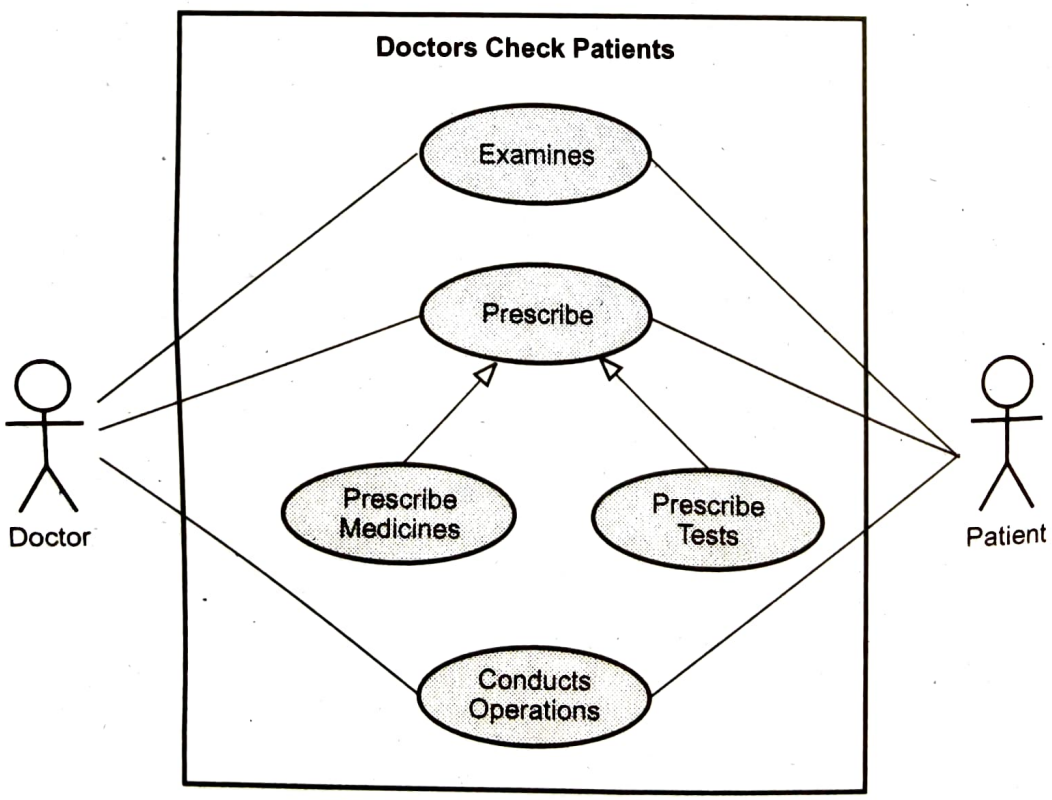


Fig. 1.16.10 (b) Use case diagram for hospital management system

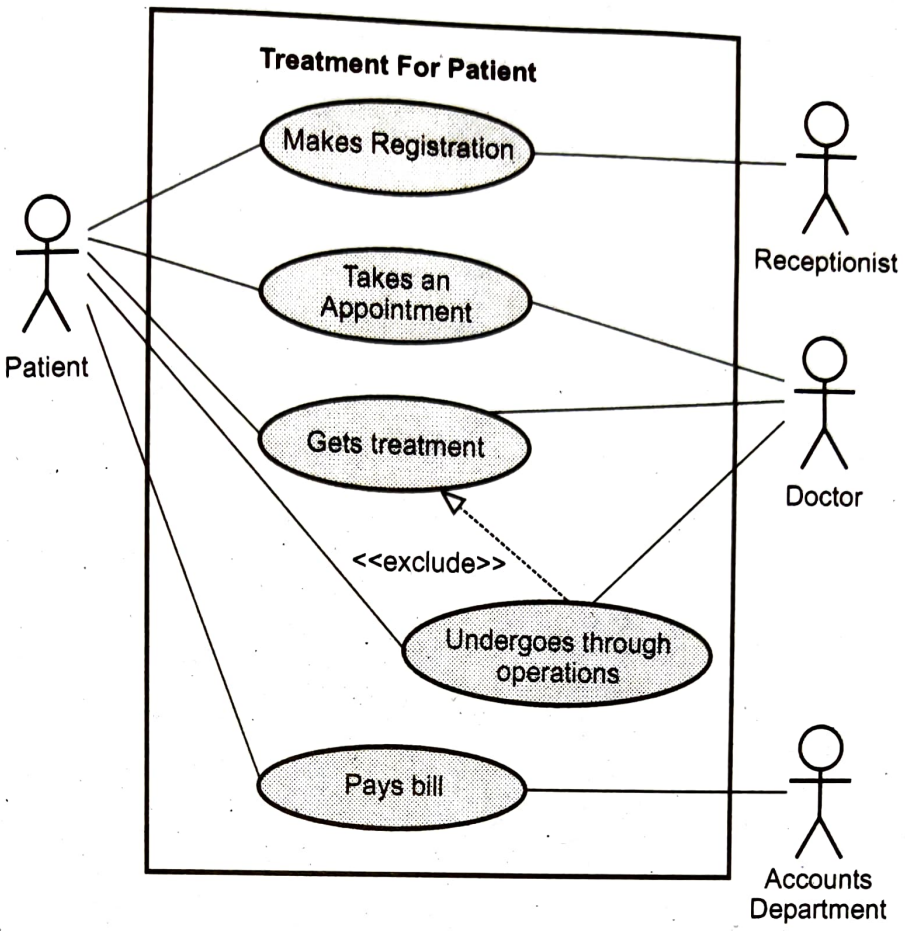


Fig. 1.16.10 (c) Hospital management system

Example 1.16.5 *Draw a use case diagram that depict the context of a credit card validation system. Explain briefly.*

Solution : (See Fig. 1.16.11 on next page.)

In the credit card validation system, there are various actors that surrounds the system. The Customer can be of Individual customer and Corporate Customer. There two more customers that are included in this transactions and those are Retail Shopkeeper and Card authorization system. Various functionalities involved are making card transaction, processing of bill, reconciling of transactions and customer account management.

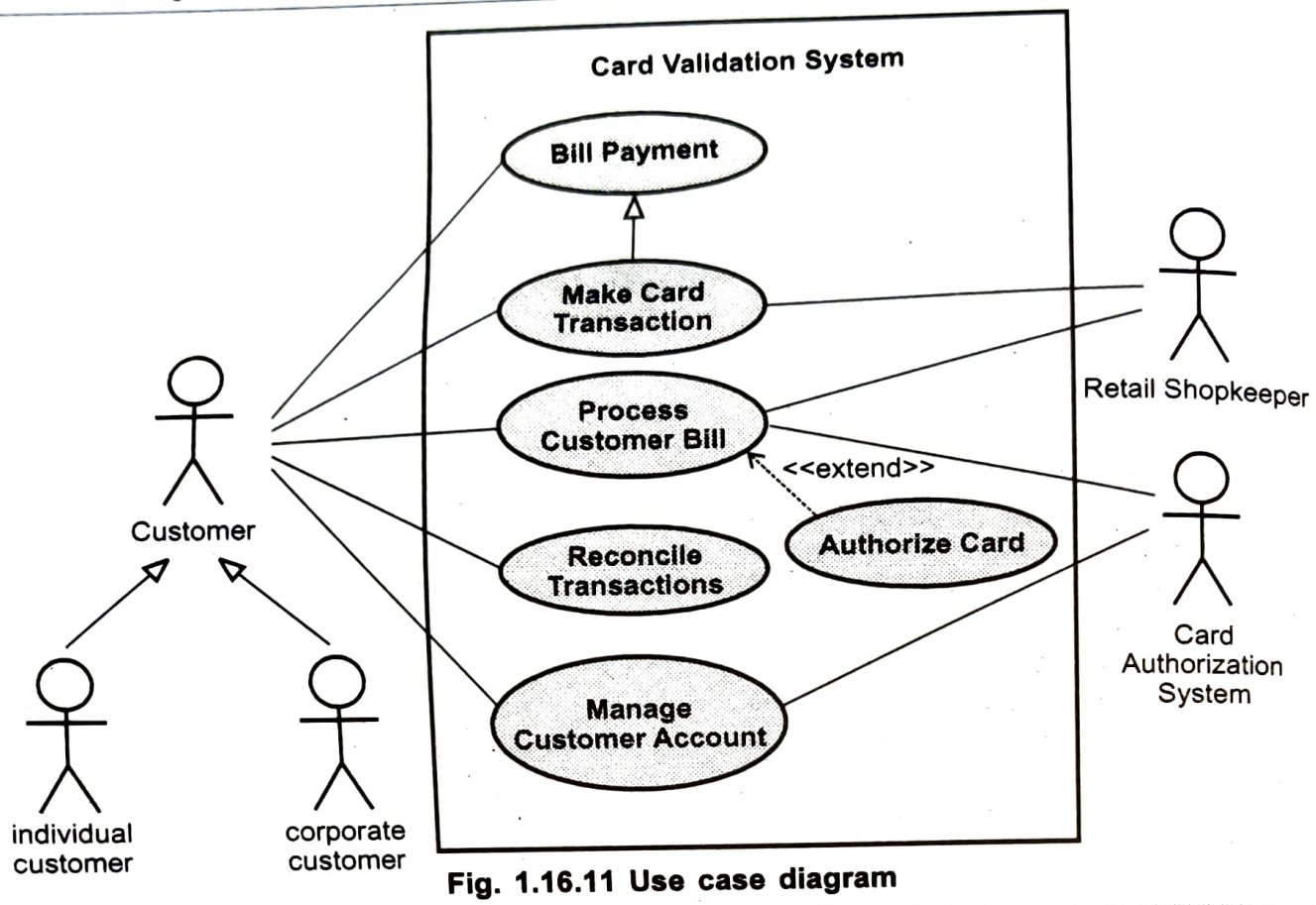


Fig. 1.16.11 Use case diagram

Example 1.16.6 *Consider an automated soda machine that gives cool drinks. Draw use case model of the soda machine.*

Solution :

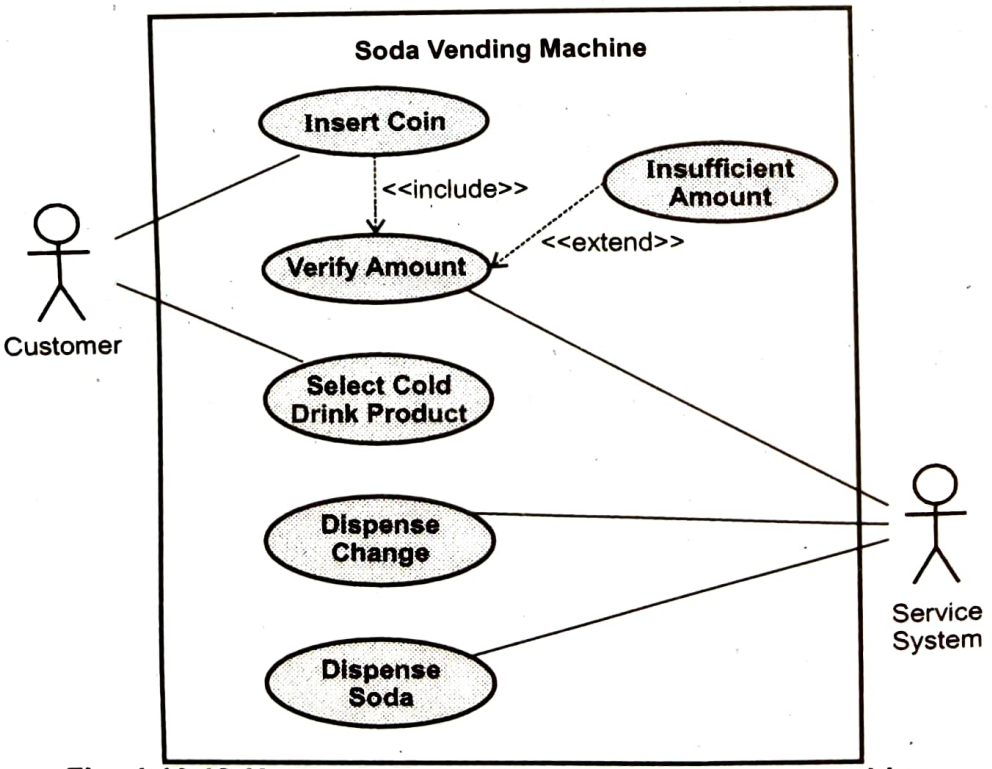


Fig. 1.16.12 Use case model for automated soda machine

Example 1.16.7 *Draw use case diagram to model the behavior of a cellular phone. Explain briefly.*

Solution : In above use case diagram, there are two important actors - user who operates the Cell phone and the Cellular network. The three important use cases are -

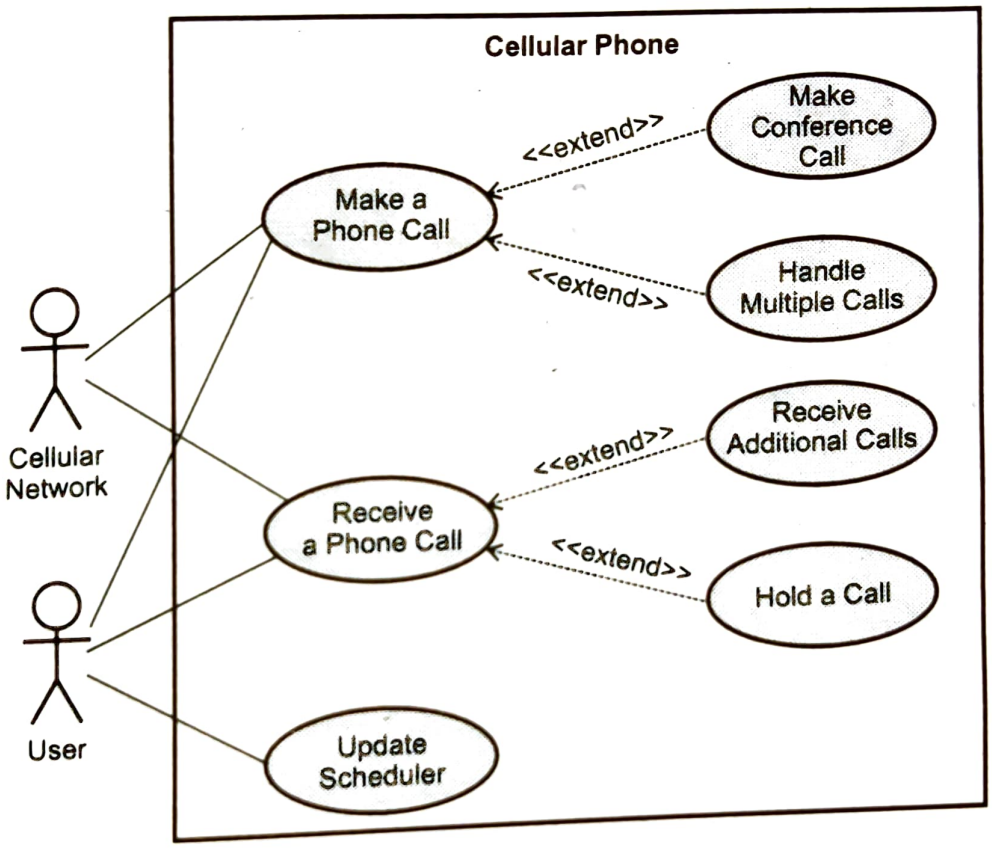


Fig. 1.16.13 Use case model for cellular phone

Use case Name : Make a phone call

In this use case the activity of user makes a call is represented. While making a call user might make a conference call for some group communication.

Use case Name : Receive a phone call

The receiving call can be attended in normal scenario. But user can receive some additional calls or he/she can hold some receiving call. These two activities are represented using the extended relationship.

Use case Name : Update scheduler

The user can update his/her schedule using calendar.

Example 1.16.8 *Design use case diagram for online shopping system.*

Solution :

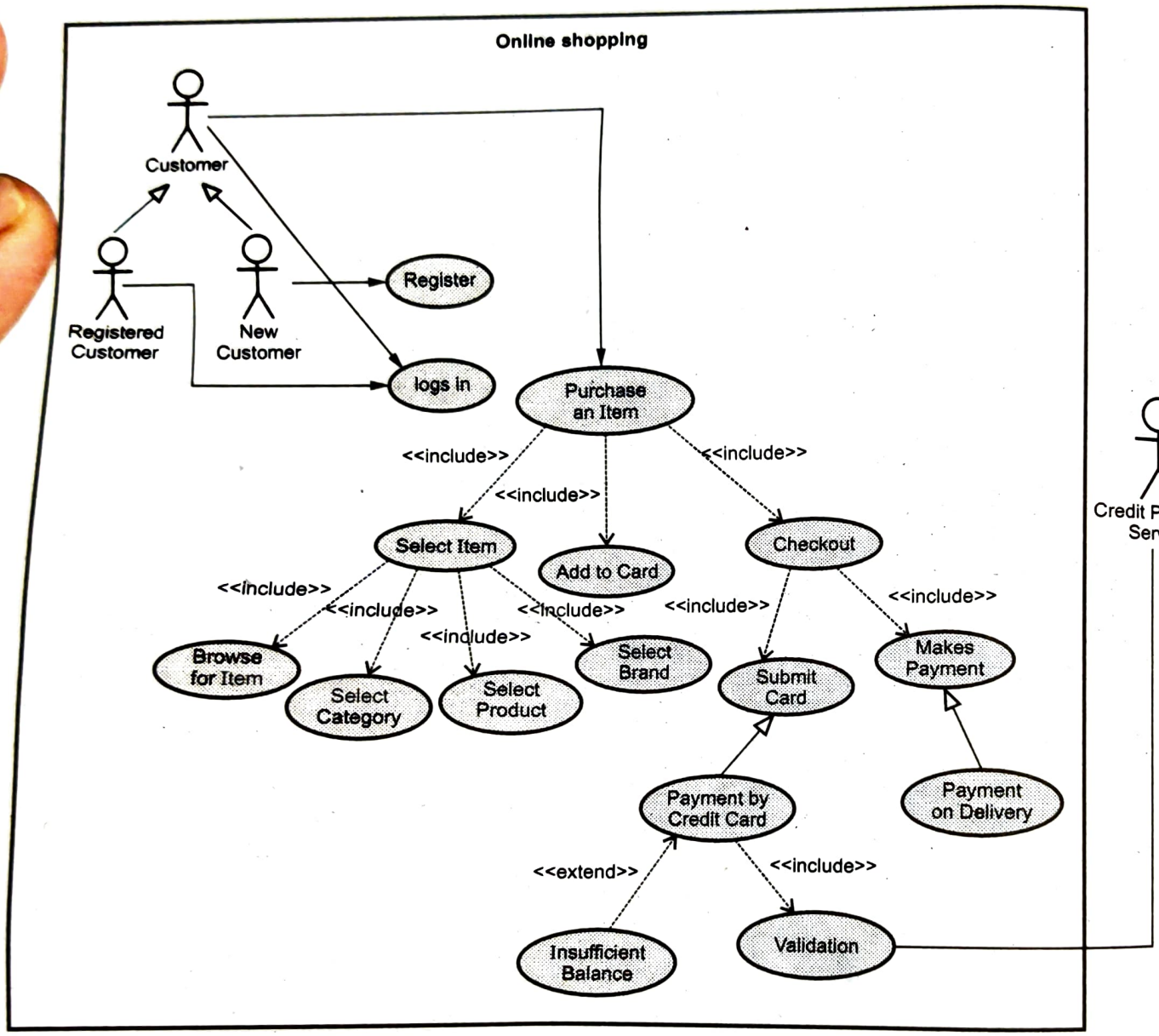


Fig. 1.16.14 Use case diagram for online shopping

Example 1.16.9 *Design use case diagram for online railway reservation system*

Solution : (See Fig. 1.16.15 on next page.)

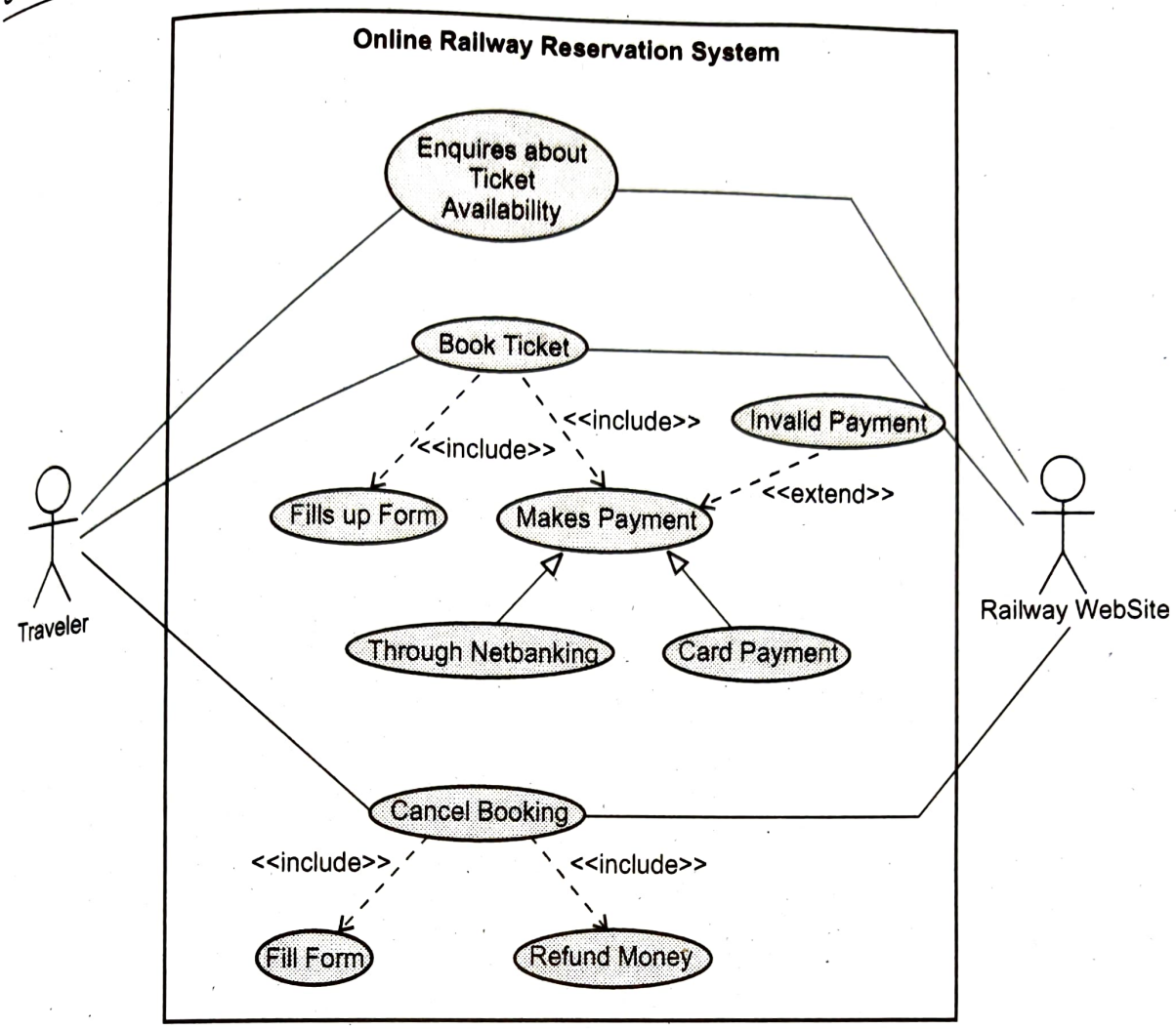


Fig. 1.16.15

Example 1.16.10 *Draw Use Case diagram for online digital library information system with all advanced notations.* **SPPU : May-17, End Sem, Marks 5**

Solution : See Fig. 1.16.16 on next page.

Example 1.16.11 *Consider software that manages electronic music files. Prepare a use case diagram and include appropriate relationships for use cases.*

a. Play a song b. Play a library c. Randomized order d. Delete a song e. Destroy a song f. Add a song. **SPPU : Aug.-15, In Sem, Marks 5**

Solution : We will first elaborate each use case. Then the use case diagram can be prepared by adding the appropriate relationship.

a. **Play a song :** Add the desired song to the end of the play list.

b. **Play a library :** Add songs to the library for playing it.

c. **Randomized order :** Randomly reorder the songs in the play list.

d. **Delete a song :** Delete a song from library.

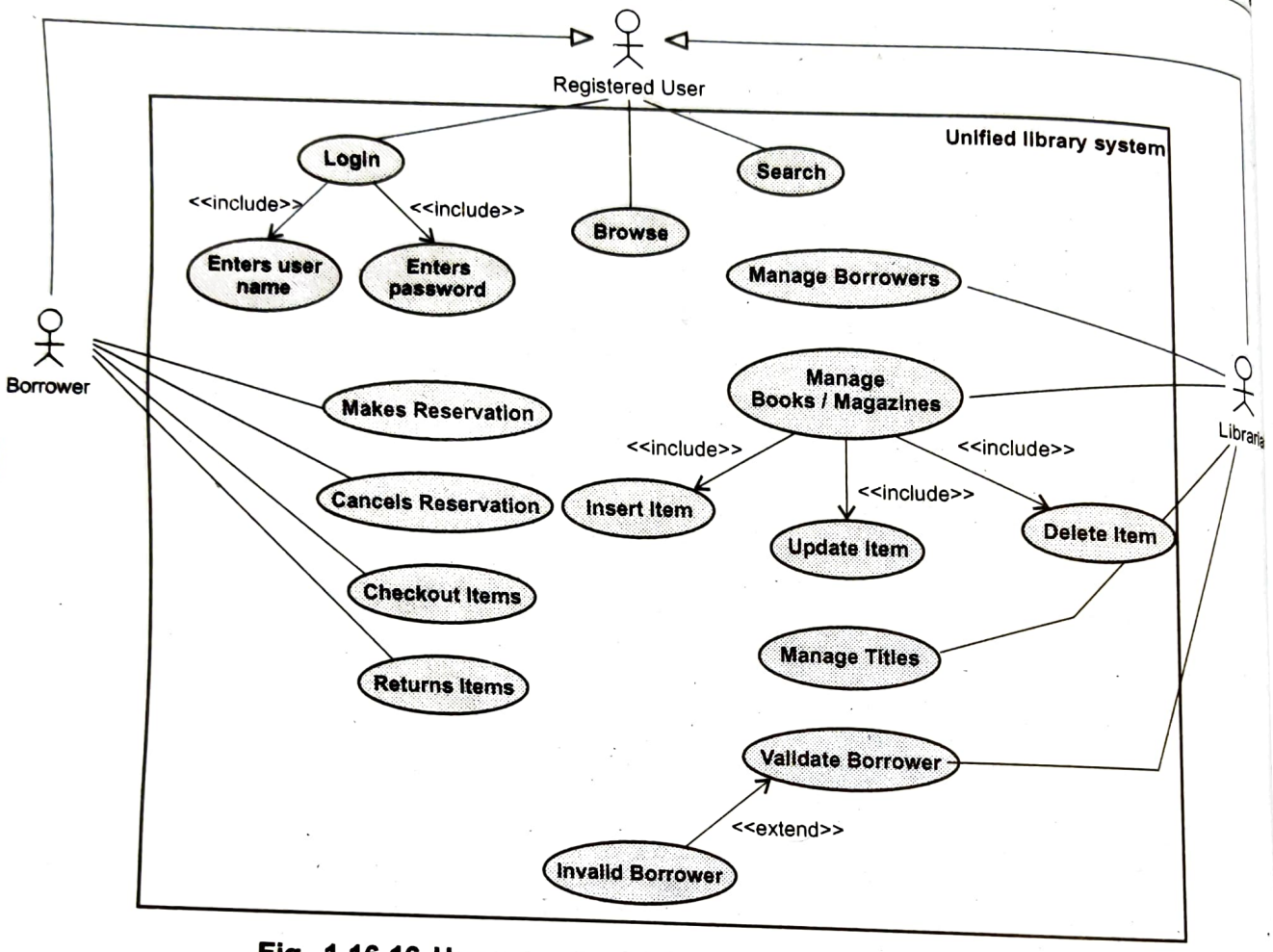


Fig. 1.16.16 Use case model for unified library application

e. Destroy a song : Delete a song from all the libraries and destroy all the file.

f. Add a song : Add a music file to the library.

Representation of these operations in use case diagram may include some other supporting operations. These operations are also represented in the following use case diagram -

Refer Fig. 1.16.17 on next page.

Example 1.16.12 *Draw a use case diagram with appropriate realtionships and notations for the following description. Consider an online travel planner software. Through this software, the user can book bus tickets, provide information, provide address, book a car on rent, book a hotel room. It is mandatory to provide payment information and provide address for booking of bus tickets, car on rent and a hotel room.*

SPPU : Dec.-16, End Sem, Marks 4

Solution : Refer Fig. 1.16.18 on page 1-50.

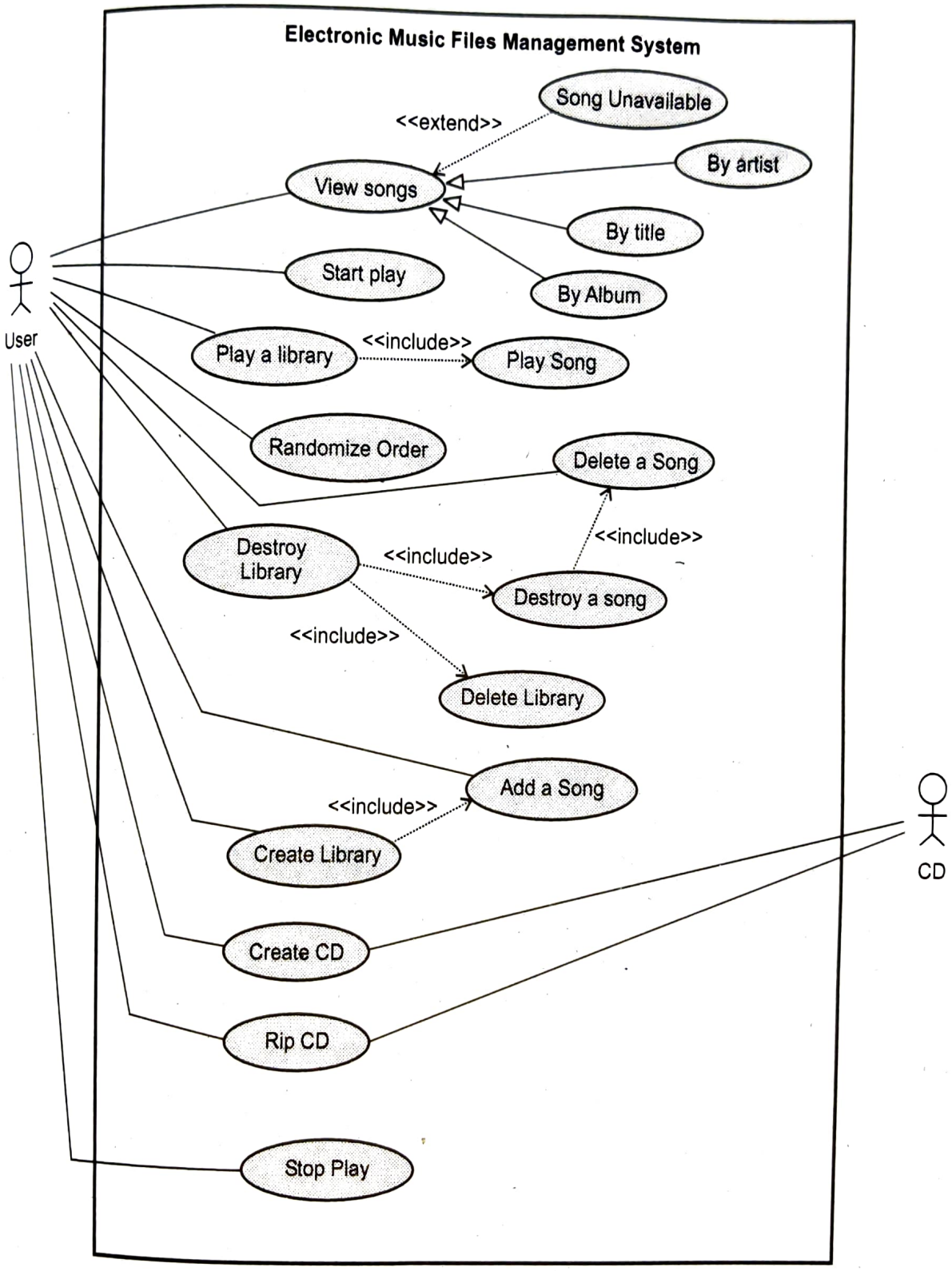

Electronic Music Files Management System
Song Unavailable
<<extend>>
By artist
View songs
By title
By Album
Start play
User
Play a library
<<include>>
Play Song
Randomize Order
Delete a Song
Destroy Library
<<include>>
<<include>>
Destroy a song
<<include>>
Delete Library
Add a Song
<<include>>
Create Library
CD
Create CD
Rip CD
Stop Play

Fig. 1.16.17

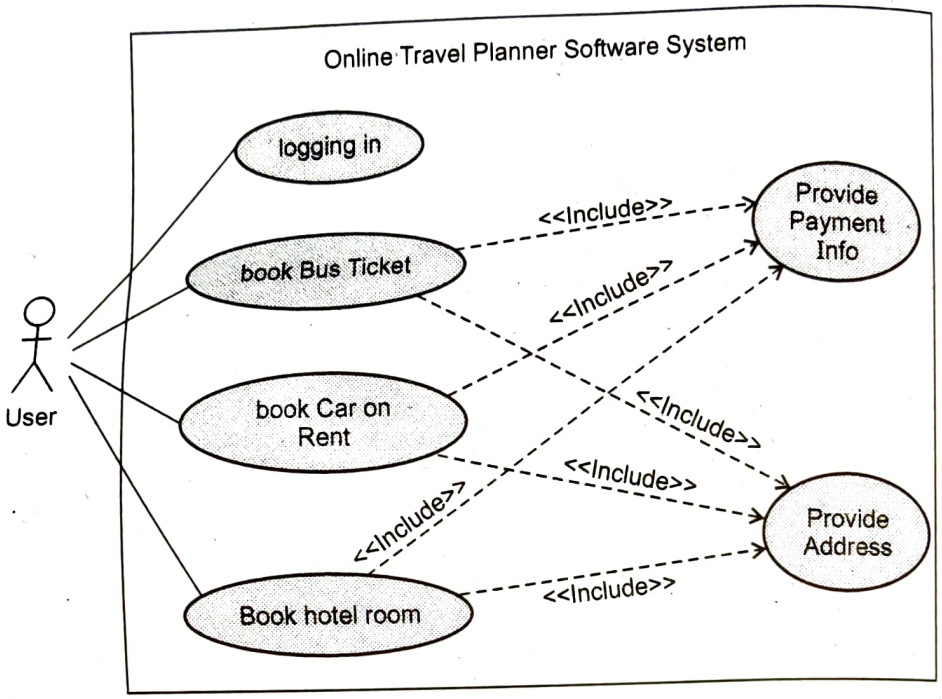


Fig. 1.16.18

Example 1.16.13 *Draw a use case diagram with appropriate relationships and notations for the following description. A software is to be developed for an alarm clock simulation. user can choose a display mode of 12 hour display or 24 hour display. User can set time, user can set alarm, turn off alarm or snooze.* **SPPU : Dec.-16, End Sem, Marks 4**

Solution :

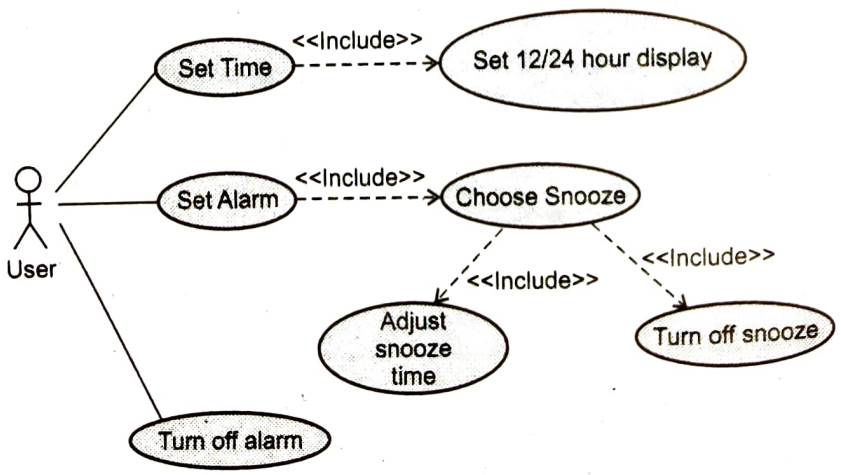


Fig. 1.16.19

Example 1.16.14 *Ticket reservation software provides following facilities to the users namely i) Check ticket availability ii) Add passenger details iii) Book ticket iv) Make payment The payment can be made by credit card or net banking. While booking a ticket, the user may optionally choose a seat of his preference. Draw a suitable use case diagram.*

SPPU : Aug.-17, Marks 5

Solution :

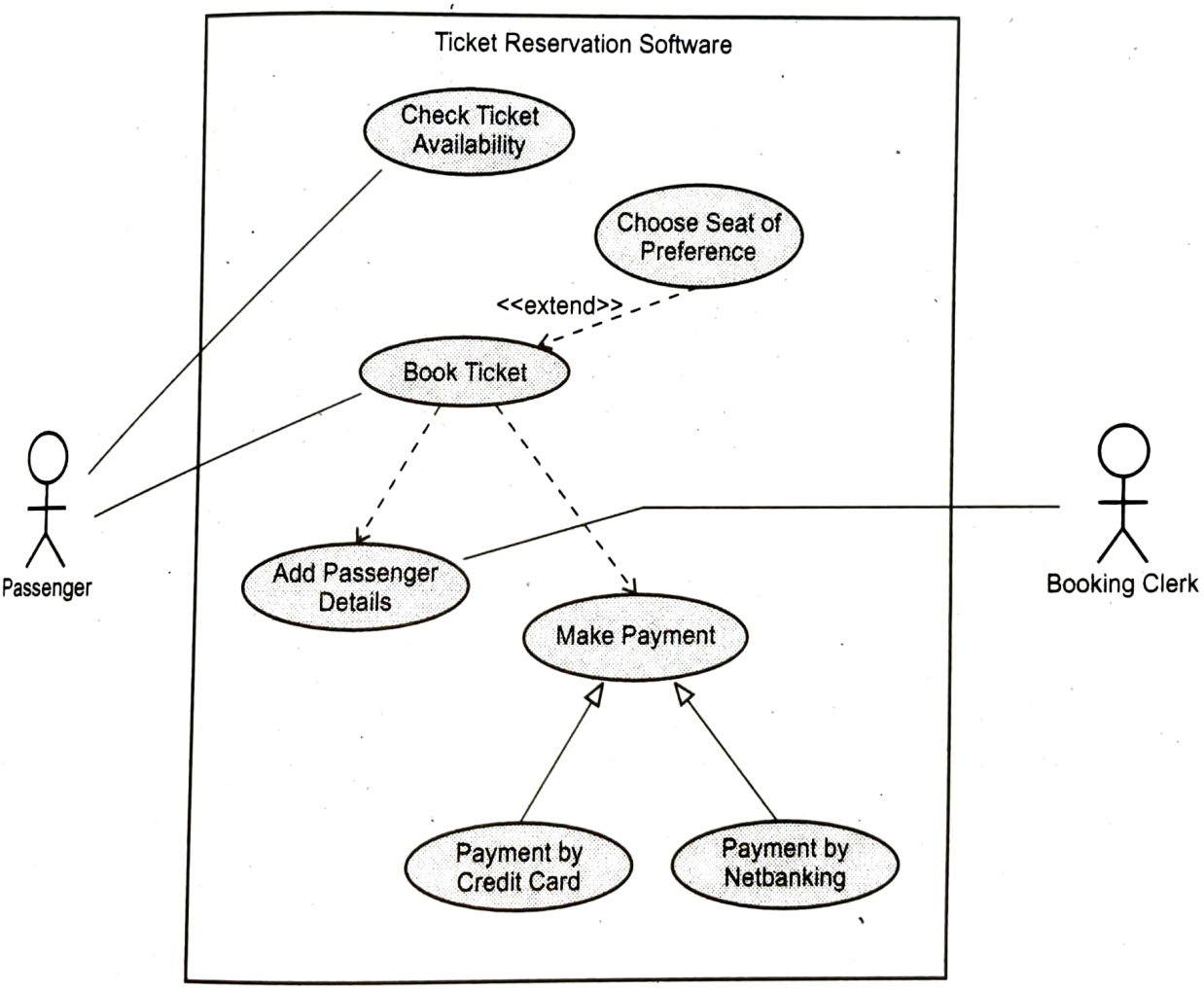


Fig. 1.16.20

Review Questions

1. *Mention the significance of extends and the includes relation in use case diagram with a suitable example.*

   SPPU : April-18, In Sem, Marks 5

2. *Explain include and extend relationship in use case model with example.*

   SPPU : March-20, In Sem, Marks 5

1.17 Use Case Template

SPPU : April-18, Dec.-18, Marks

- During requirement gathering the primary requirements of the system understood. These basic requirements of the system are called the **functio requirements**.
- In order to understand the working of these functional requirements, developers and users create **a set of scenarios**. These scenarios are represented use cases.
- The use detailed description is represented in text form. Following table shows template of the use case -

Use Case Description	
Use case Name :	Write name of use case
Scenario :	Write the name of the scenario
Triggering Event :	The triggering event due to which the function starts, is described.
Brief Description :	Brief description of the use case is written here.
Actors :	Specify the role of the entity who interacts with the system.
Related Use Cases :	Other related use cases that co-exist along with this use case.
Stakeholders :	Other than primary actor, there are other entities that interact with th system. These entities are called stakeholders.
Preconditions :	The condition that is to be satisfied before the use case start is specifi as precondition.
Postconditions :	The condition that is to be satisfied after the use case execution is specified as postcondition.
Flow of Events :	Flow of activities that occur during the execution of this use case.
Exception Conditions :	The exceptional situations that may occur during the execution of th case is described here.

Example 1.17.1 *Write the use case description for ATM System*

Solution : An informal description or text story for withdrawal of money from ATM is as follows -

- "Customer wants to operate the ATM machine for performing some transactions. For instance customer wants to withdraw the money. He enters the ATM card and then types in the PIN. The system validates the customer. If the customer is the valid customer then the customer is allowed to do further transactions. Otherwise the card will be rejected. The valid customer enters the amount to be withdrawn. It is then checked if withdrawal amount < balance amount. If it is so, then the machine dispenses the desired amount. If there is no further transactions then the card is ejected. Customer collects the cash, statement and card."

Use Case Description		
Use case Name :	Withdraw money from ATM System	
Scenario :	To monitor the withdrawal of money from ATM	
Triggering Event :	Customer enters the pin and request for withdrawal of money.	
Brief Description :	Customer requests for withdrawal of money by entering the amount to be withdrawn. If the sufficient balance is present in the account then, the money is dispensed from the ATM machine.	
Actors :	Customer	
Related Use Cases :	Check balance, get_Statement	
Stakeholders :	Administrator	
Preconditions :	ATM system has to be programmed and as to recognize and validate the PIN.	
Postconditions :	The requested amount must be ejected, card must be ejected and balance must be updated. The customer logs off.	
Flow of Events :	**Actor**	**System**
	1. Customer observes the control panel.	1. The system displays the Welcome Screen
	2. Customer enters the ATM card	2. System validates the card.

	3. Customer enters the PIN.	3. System validates the PIN.
	4. Customer selects the operation(withdrawal, deposit, inquiry)	4. System display main menu and allows the use to select one option.
	5. Customer collects the cash, statement, card etc.	5. System displays the payment details screen.
		6. System beeps on ejecting money and card.
Exception Conditions :	1. The control panel is not ready. 2. PIN is incorrect. 3. Card is not recognized. 4. Insufficient balance 5. Limit for the transaction exceeds. 6. Total number of allowed transactions per day.	

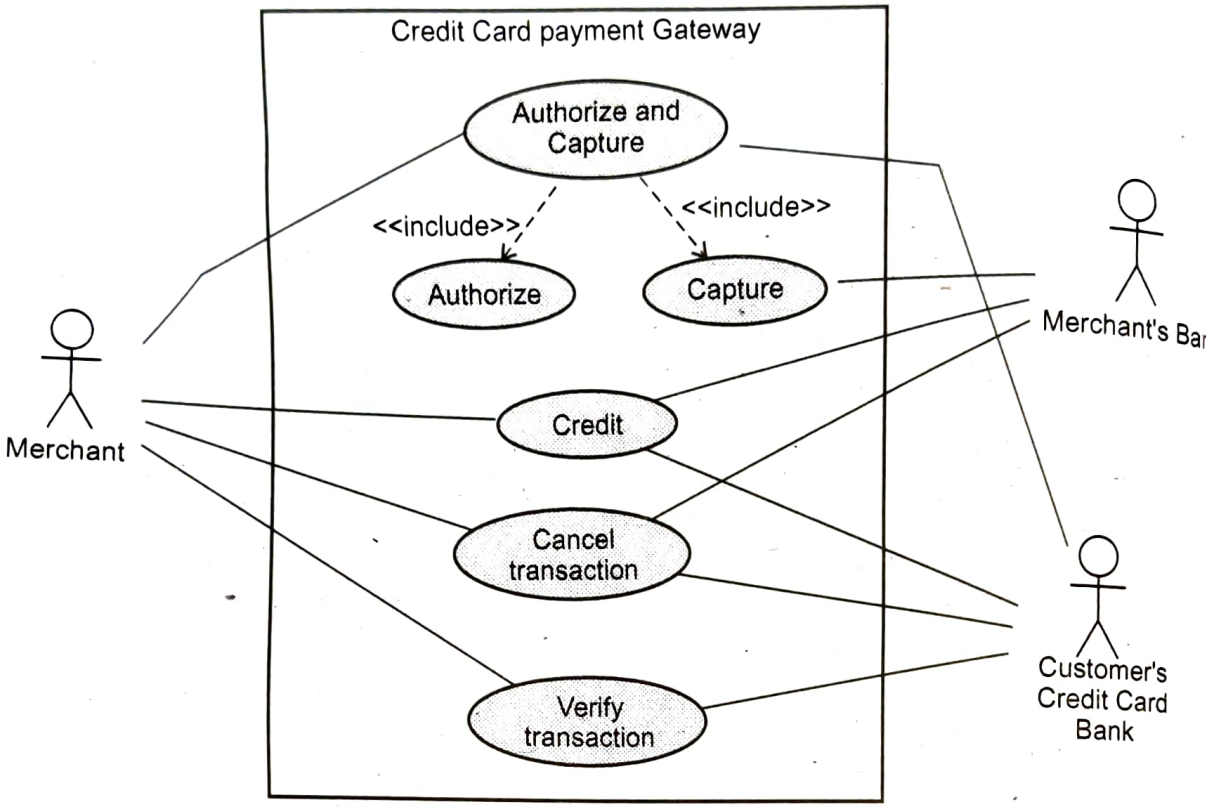
- Another use case description is for **deposit money** operation.

Use Case Description	
Use case Name :	Deposit money through ATM System
Scenario :	To monitor the deposition of money from ATM
Triggering Event :	Customer enters the pin and request for deposition of money.
Brief Description :	Customer requests for deposition of money in his account via ATM machine.
Actors :	Customer
Related Use Cases :	Check balance, get_Statement
Stakeholders :	Administrator
Preconditions :	ATM system has to be programmed and as to recognize and validate the PIN.

Postconditions :	The customer deposits money and logs off.	
Flow of Events :	**Actor**	**System**
	1. Customer observes the control panel.	1. The system displays the Welcome Screen
	2. Customer enters the ATM card	2. System validates the card.
	3. Customer enters the PIN.	3. System validates the PIN.
	4. Customer selects the deposit money option.	4. System display main menu and allows the use to select one option.
	5. Customer enters the amount to be deposited.	5. System displays the message to place the depositing amount on the slot.
	6. Customer places money on deposit slot.	6. System accepts the money and checks the exact amount.
	7. Customer collects the card etc.	7. System beeps on ejecting card.
Exception Conditions :	1. The control panel is not ready. 2. PIN is incorrect. 3. Card is not recognized. 4. Limit for the transaction exceeds. 5. Non positive amount is entered to deposit the money. 6. Failure to place the depositing amount on depositing slot. 7. Total number of allowed transactions per day.	

Example 1.17.2 *Draw a use case diagram for the system. Credit card authentication system. The scenario is : The merchant submits a credit card trasnaction request to the credit card payment gateway on behalf of a customer. Bank which issued customer's credit card is actor which could approve or reject the transaction. If transaction is approved, funds will be transferred to merchant's bank account.* **SPPU : April-18, In Sem, Marks 5**

Solution :



Example 1.17.3 *Write a use case specification for the use case : User authentication.*

SPPU : Dec.-18, End Sem, Marks

Solution :

Use Case :	User Authentication
Primary Actor :	User
Scenario :	
Precondition :	ATM has detected a card in a slot.
Flow of Events :	1. Customer slides card in and out.
	2. Machine prompts the user for password.
	3. Customer enters a password.
	4. If password is in-correct
	a) If card stolen generate acknowledgement for branch.

	b) Go back to step 2 c) If password is incorrect for three times d) Retain card and notify user. 5. System authenticates user.
Non Functional Requirement :	Authentication should take place within 20 seconds after password entered.
Assumption :	Customer speaks English

1.18 Multiple Choice Questions

Q.1 The object oriented modeling for building system takes _____ as the basis

a class b object

c model d function

Q.2 During the design phase, the overall _____ of the system is described.

a architecture b system flow

c data flow d none of the above

Q.3 In object oriented development life cycle is composed of ____

a analysis, design, and implementation steps in the given order and using multiple iterations.

b analysis, design, and implementation steps in the given order and going through the steps no more than one time.

c analysis, design, and implementation steps in any order and using multiple iterations.

d analysis, design, and implementation steps in any order and going through the steps no more than one time.

Q.4 Grady Booch, James Rumbaugh, and Ivar Jacobson combined the best features of their individual object-oriented analysis into a new method for object oriented design known as _____.

a HTML b XML

c UML d SGML

Q.5 A design description in OOD includes _____.

a protocol description
b implementation description
c type description
d both a and b

Q.6 MDA is an approach of producing_____ from modeling diagrams

a analysis
b design
c code
d none of these

Q.7 The ___ is a graphical modelling language that provides us with syntax describing the major elements of software systems.

a UML
b flowchart
c data flow diagrams
d none of these

Q.8 Which things in UML are the explanatory parts of UML models ?

a Grouping things
b Structural things
c Behavioural things
d Annotational things

Q.9 Which of the following is not an UML diagram ?

a Class diagram
b Object diagram
c Interface diagram
d Use case Model

Q.10 An UML diagram that facilitates the requirements gathering and interacts betwe the system and external users is called as ____.

a flow chart diagram
b use case diagrams
c class Diagram
d state-transition diagram

Q.11 An UML diagram which has a static view is _____.

a Use case
b Class diagram
c List
d None of these

Q.12 Use case descriptions consist of interaction among which of the following ?

a Product
b Use case
c Actor
d Product and Actor

Q.13 What are the notations for the use case diagrams ?

a Use case
b Actor
c Prototype
d Use case and actor

Q.14 Which among the following can be heuristic for use case diagram ?

a The product can be made actor
b Never name actors with noun phrases
c Name Use cases with verb phrases
d All of the mentioned

Q.15 The UML diagram that shows the interaction between users and system is called as _____.

a class diagram
b object diagram
c use case diagram
d component diagram

Q.16 COMET stands for _____.

a Common Object Modeling and Design Method
b Collaborative Object Modeling and Design Method
c Collective Modeling and Engineering Technique
d Collaborative Modeling and Enhancement Technique

Q.17 COMET is a _____.

a requirements engineering technique
b analysis technique
c design technique
d implementation methodology

Q.18 Which one of the following is not a step of requirement engineering ?

a Elicitation
b Design
c Analysis
d Documentation

Q.19 What is the first step of requirement elicitation ?

a Identifying Stakeholder
b Listing out Requirements
c Requirements Gathering
d All of these

Q.20 The user system requirements are the parts of which document ?

a SDD

b SRS

c DDD

d SRD

Answer Keys for Multiple Choice Questions :

Q.1	b	**Q.2**	a	**Q.3**	a	**Q.4**	c
Q.5	d	**Q.6**	c	**Q.7**	a	**Q.8**	d
Q.9	c	**Q.10**	b	**Q.11**	a	**Q.12**	d
Q.13	d	**Q.14**	c	**Q.15**	c	**Q.16**	b
Q.17	c	**Q.18**	b	**Q.19**	a	**Q.20**	b