UNIT 1 INTRODUCTION TO OBJECT ORIENTED MODELING

Structure **Page Nos.**

1.0 INTRODUCTION

Object oriented design methods emerged in the 1980s, and object oriented analysis methods emerged during the 1990s. In the early stage, object orientation was largely associated with the development of graphical user interfaces (GUIs), and a few other applications became widely known. In the 1980s, Grady Booch published a paper on how to design for Ada, and gave it the title, *Object Oriented Design*. In 1991, Booch was able to extend his ideas to a genuinely object oriented design method in his book with the same title, revised in 1993 (Booch, 1994) [*sic*].

The Object Modeling Technique (OMT) covers aspects of object oriented analysis and design. OOT provides a very productive and practical way of Software development. As object oriented Technology (OOT) is not language dependent, there is no need for considering a final implementation language, during Object Oriented Modeling (OOM). OOT combine structural, control and functional aspects of the system. We will discuss the structural, control and functional aspects of the systems in great detail in block 3 of this course.

In this unit, we will discuss the basic notions of object orientation. This unit will cover discussion on objects, classes, links, association, generalization, and inheritance. We will discuss the basics of an object model with the help of an example. Towards the end of this unit we will cover the benefits of OOM. In this unit, you will also be introduced to some OOAD tools.

1.1 OBJECTIVES

After going through this unit, you should be able to:

- explain basics of object oriented Modeling;
- define Objects and Classes;
- explain the concepts of links and Associations;
- explain the concept of Generalization and Inheritance;
- describe benefits of Object Oriented Modeling, and
- explain the use of some OOAD tools.

1.2 OBJECT ORIENTED MODELING

Object oriented modeling is entirely a new way of thinking about problems. This methodology is all about visualizing the things by using models organized around real world concepts. Object oriented models help in understanding problems, communicating with experts from a distance, modeling enterprises, and designing programs and database. We all can agree that developing a model for a software system, prior to its development or transformation, is as essential as having a blueprint for large building essential for its construction. Object oriented models are represented by diagrams. A good model always helps communication among project teams, and to assure architectural soundness.

It is important to note that with the increasing complexity of systems, importance of modeling techniques increases. Because of its characteristics Object Oriented Modeling is a suitable modeling technique for handling a complex system. OOM basically is building a model of an application, which includes implementation details of the system, during design of the system.

Brooks observes that the hard part of software development is the manipulation of its essence due to the inherent complexity of the problem, rather than the accidents of its mapping into a particular language, which are due to temporary imperfections in our tools that are rapidly being corrected (Brooks-87).

As you know, any system development refers to the initial portion of the software life cycle: analysis, design, and implementation. During object oriented modeling identification and organization of application with respect to its domain is done, rather than their final representation in any specific programming language. We can say that OOM is not language specific. Once modeling is done for an application, it can be implemented in any suitable programming language available.

OOM approach is a encouraging approach in which software developers have to think in terms of the application domain through most of the software engineering life cycle. In this process, the developer is forced to identify the inherent concepts of the application. First, developer organize, and understood the system properly and then finally the details of data structure and functions are addressed effectively.

In OOM the modeling passes through the following processes:

- System Analysis
- System Design
- Object Design, and
- Final Implementation.

System Analysis: In this stage a statement of the problem is formulated and a model is build by the analyst in encouraging real-world situation. This phase show the important properties associated with the situation. Actually, the analysis model is a concise, precise abstraction and agreement on how the desired system must be developed. You can say that, here the objective is to provide a model that can be understood and criticized by any application experts in the area whether the expert is a programmer or not.

System Design: At this stage, the complete system architecture is designed. This is the stage where the whole system is divided into subsystems, based on both the system analysis model and the proposed architecture of the system.

Object Design: At this stage, a design model is developed based on the analysis model which is already developed in the earlier phase of development. The object design decides the data structures and algorithms needed to implement each of the classes in the system with the help of implementation details given in the analysis model.

Final Implementation: At this stage, the final implementation of classes and relationships developed during object design takes place a particular programming language, database, or hardware implementation (if needed). Actual implementation

should be done using software engineering practice. This helps to develop a flexible and extensible system.

Whole object oriented modeling is covered by using three kinds of models for a system description. These models are:

- object model,
- dynamic model, and
- functional model.

Object models are used for describing the objects in the system and their relationship among each other in the system. The dynamic model describes interaction among objects and information flow in the system. The data transformations in the system are described by a functional model. **All three models are applicable during all stages of development**. These models bear the responsibility of acquiring implementation details of the system development. It is important to note that you cannot describe a system completely until unless all three modes are described properly. In block 3 of this course, we will discuss these three models in detail.

Before we discuss the characteristics of object oriented modeling, let us see how object oriented development is different from structured development of the system. In the structured approach, the main emphasis is **on specifying and decomposing system functionality**. Structured approach is seen as the **most direct way of implementing a desired goal**. A structured approach has certain basic problems, such as, if the requirements of system change then a system based on decomposing functionality may require massive restructuring, and, the system gradually become unmanageable. In contrast to the structured approach, the basic focus of object-oriented approach is to **identify objects** from the application domain, and then to **associate procedures** (methods) around these identified objects.

You can say that object oriented development is an indirect way of system development because in this approach **a holistic view of application domain is considered, and objects are identified** in the related problem domain. A historic view of application helps in realizing the situations and characteristics of the system. Taking a **holistic view of the problem domain rather than considering functional requirements of a single problem give an edge to object oriented development**. Once the objects are created with the needed characteristics, they communicate with each other **by message passing** during problem solving.

☞ Check Your Progress 1

1) What is OOM?

……..

……..

……..

2) List different steps involved in OOM process.

……..

……..

……..

3) Differentiate OO development from structured development.

……..

……..

……..

1.3 BASIC PHILOSOPHY OF OBJECT ORIENTATION

There are several characteristics of object-oriented technology. Some of these characteristics have been discussed in course MCS-024. We have also implemented some of them in Java programming language, although these characteristics are not unique to object-oriented systems in the sense that they vary from object based systems to object oriented systems. However, most of the properties are particularly well supported in object oriented systems.

Now, let us discuss about the basic characteristics around which object oriented systems are developed.

Abstraction

Abstraction is one of the very important concepts of object oriented systems Abstraction focues on the essential, inherent aspects of an object of the system. It does not represent the accidental properties of the system. In system development, abstraction helps to focus on what an object is supposed to do, before deciding how it should be implemented. The use of abstraction protects the freedom to make decisions for as long as possible, by avoiding intermediate commitments in problem solving. Most of the modern languages provide data abstraction. With the abstraction, ability to use inheritance and ability to apply polymorphism provides additional freedom and capability for system development. When you are using abstraction during analysis, you have to deal with application-domain concepts. You do not have to design and make implementation decisions at that point.

Encapsulation

Encapsulation, or information hiding, is the feature of separating the external aspects of an object, from the internal implementation details of that object. It helps in hiding the actual implementation of characteristics of objects. You can say that encapsulation is hiding part of implementation that do internal things, and these hidden parts are not concerned to outside world. Encapsulation enables you to combine data structure and behaviour in a single entity. Encapsulation helps in system enhancement. If there is a need to change the implementation of object without affecting its external nature, encapsulation is of great help.

Polymorphism

Class hierarchy is the deciding factor in the case of more than one implementation of characteristics. An object oriented program to calculate the area of different Figures would simply call the Find_ Area operation on each figure whether it is a circle, triangle, or something else. The decision of which procedure to use is made implicitly by each object, based on its class polymorphism makes maintenance easier because the calling code need not be modified when a new class is added.

Sharing of Structure and Behaviour

One of the reasons for the popularity of object-oriented techniques is that they encourage sharing at different levels. Inheritance of both data structure and behaviour allows common structure (base class) to be used in designing many subclasses based on basic characteristics of base class, and develop new classes with less effort. Inheritance is one of the main advantages of any object oriented language, because it gives scope to share basic code.

In a broader way we can say that object oriented development not only allows information sharing and reuse within an application, but also, it gives a base for project enhancement in future. As and when there is a need for adding new characteristics in the system, they can be added as an extension of existing basic

features. This can be done by using inheritance, and that too, without major modification in the existing code. But be aware that just by using object orientation you do not get a license to ensure reusability and enhancement. For ensuring reusability and enhancement you have to have a more general design of the system. This type of design can be developed only if the system is properly studied and features of proposed system are explored.

Emphasis on Object Structure, not on Operation Implementation

In object orientation the major emphasis is on specifying the characteristics of the objects in a system, rather than implementing these characteristics. The uses of an object depend highly on the facts of the application and regular changes during development. As requirements extend, the features supplied by an object are much more stable than the ways in which they are used, hence software systems built on object structure are more secure.

While developing a system using the object oriented approach, main emphasis is on the essential properties of the objects involved in the system than on the procedure structure to be used for implementation. During this process what an object is, and its role in system is deeply thought about.

1.4 CHARACTERISTICS OF OBJECT ORIENTED MODELING

In object oriented modeling objects and their characteristics are described. In any system, objects come into existence for playing some role. In the process of defining the roles of objects, some features of object orientation are used. In this section we will discuss these features, which include:

- Class and Objects
- Links and Association
- Generalization and Inheritance

Let us start our discussion with Class and Objects.

1.4.1 Class and Objects

A *class* is a collection of things, or concepts that have the same characteristics. Each of these things, or concepts is called an *object.*

We will discuss, in the next unit of this block, that the class is the most fundamental construct within the UML.

Classes define the basic words of the **system being modeled**. Using a set of classes as the core vocabulary of a software project tends to greatly facilitate understanding and agreement about the meanings of terms, and other characteristics of the objects in the system.

Classes can serve as the foundation for data modeling. In OOM, the term classes is usually the base from which visual modeling tools−such as Rational Rose XDE, Visual Paradigm function and design the model of systems.

Now, let us see how the characteristics that classes share are captured as attributes and operations. These terms are defined as follows:

- *Attributes* are named slots for data values that belong to the class. As we have studied in MCS-024, different objects of a given class typically have at least some differences in the values of their attributes.
- *Operations* represent services that an object can request to affect the behaviour of the object or the system itself.

In unit 3 of this block, we will cover the standard UML notation for OOM in detail. Here, we will mention about standard notation of class. The notation for a class is **a box with three sections**. The top section contains the name of the class in boldface type, the middle section contains the attributes that belong to the class, and the bottom section contains the class's operations as you can see in *Figure 1.*

Figure 1: Class notation

You can, also show a class without its attributes or its operations, or the name of the class can appear by itself as shown in *Figure 2.*

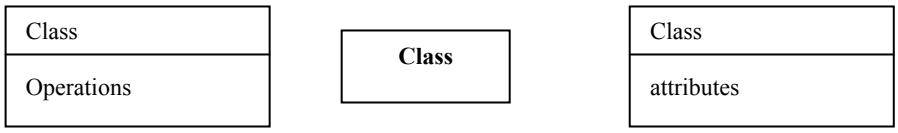

Figure 2: Alternate class notations

The naming convention for classes are as follow:

- Class names are simple nouns or noun phrases.
- Attribute names in a class are simple nouns or noun phrases. The first word is not capitalized, but subsequent words may be capital.
- Operation names are simple verbs. As with attributes, the first word is not capitalized and subsequent words may be capital.

Objects

The notation for an object is the same in basic form as that for a class. There are three differences between the notations, which are:

- Within the top section of the class box, the name of the class to which the object belongs appears after a **colon**. The object may have a name, which appears before the colon, or it may be anonymous, in which case nothing appears before the colon.
- The contents of the top compartment are underlined for an object.
- Each attribute defined for the given class has a specific value for each object that belongs to that class.

You can see the notion of an object you can see in *Figure 3.*

Figure 3*:* Notation of object

If you look around you will find many examples of real world objects such as your books, your desk, your television, etc.

Everything that the software object knows (state) and can do (behaviour) is expressed by the variables and the methods within that object. In other words, all the objects share states and behaviour. Let us say that a software object that models your real-world bicycle would have variables that indicated the bicycle's current state: its speed is 20 mph, and its current gear is the 3rd gear, etc.

Communication by Message Passing

You will agree that a single object alone is generally not very useful. Objects usually appear as a components of a larger program or a system. Through the interaction of these objects, functionality of systems are achieved. Software objects interact and communicate with each other by *message passing* to each other. When object X wants object Y to perform one of methods of object Y, object X sends a message to object Y. Message passing provide two significant benefits:

- An object's characteristics are expressed through its methods, so message passing supports all possible interactions between objects.
- It closes the gap between objects. Objects do not need to be in the same process, or even on the same machine, to send and receive messages back and forth to each other.

1.4.2 Links and Association

Links and associations are the basic means used for establishing relationships among objects and classes of the system. In the next subsection we will discuss links and associations which are used for representing relationship.

General Concepts

A link is a physical or conceptual connection between objects for example, a student, **Ravi study in IGNOU**. Mathematically, you can define a link as a tuple that is an ordered list of objects. Further, a link is also defined as an instance of an association. In other words you can say that an association is a group of links with a common structure and common meanings, for example, a student study in a university. All the links in an association connects objects from the same classes. A link is used to show a relationship between two (or more) objects.

Association and classes are similar in the sense that classes describe objects, and association describe **links**. *Figure 4a* shows us how we can show the association between Student and University

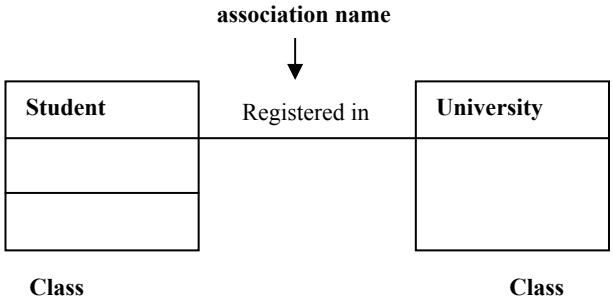

Figure 4a: Association

Note that every association has roles. For example, in *Figure 4b* you can see that two classes, Student and University, have their defined roles. Here you can also see that binary association has two roles, one from each class.

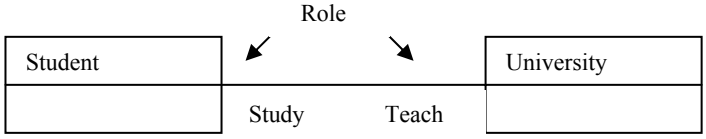

Figure 4b: Roles in association

Associations may be binary, ternary, or have higher order. In exercise, the vast majority of association are binary or ternary associations. But a ternary association is formed compulsion; they cannot be converted into binary association. If a ternary association is decomposed in some other association, some information will be lost. In *Figure 5* you can see a ternary association.

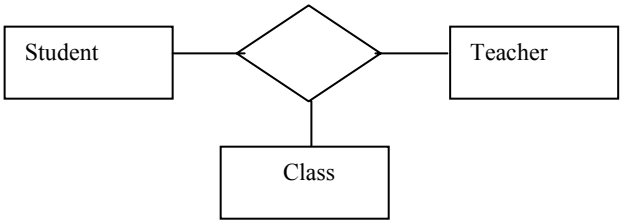

Figure 5: Ternary association

Multiplicity

Multiplicity in an association specifies how many objects participate in a relationship. Multiplicity decides the number of related objects. Multiplicity is generally explained as "one" or "many," but in general it is a subset of the non-negative integers.

Table 1: Multiplicity Indicators.

Indicator	Meaning
0..1	Zero or one
1	One only
0..*	Zero or more
1..*	One or more
n	Only *n* (where *n* > 1)
0..n	Zero to *n* (where *n* > 1)
1..n	One to *n* (where *n* > 1)

In associations, generally movement is in both the directions of the relationships but if you want to be specific in any particular direction, you have to mark it by an arrow as given in *Figure 6.*

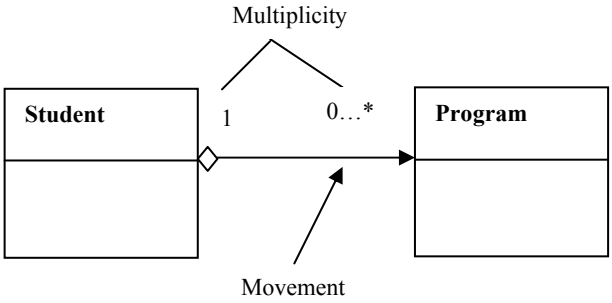

Figure 6: Association and movement

Aggregation

Aggregation is a special form of association, which models the "part-whole" or "a-part-of" relationship as an aggregate (the whole) and parts. The most considerable property of aggregation is transitivity, that is, if X is part of Y and Y is part of Z, then X is part of Z. Aggregation is seen as a relationship in which an assembly class is related to component class. In this component objects are not having separate existence, they depend on composite objects as you can see in *Figure 7* Exam Schedule is not having separate existence.

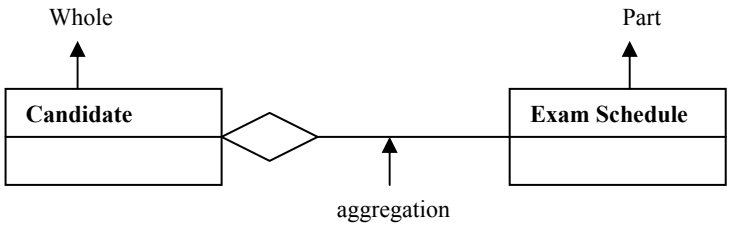

Figure 7: Association and whole-part relationship

☞ Check Your Progress 2

1) What is abstraction?

……

……

2) What is association? Give example of association.

……

……..

3) What is multiplicity in associations? Give example to explain multiplicity?

……

……

1.4.3 Generalization and Inheritance

In this section we will discuss the concepts of generalization, inheritance, and their uses in OOM.

Generalization

Generalization and inheritance are powerful abstractions for sharing the structure and/or behaviour of one or more classes.

Generalization is the relationship between a class, and it defines a hierarchy of abstraction in which subclasses (one or more) inherit from one or more superclasses.

Generalization and inheritance are transitive across a subjective number of levels in the hierarchy. Generalization is an "is-a-kind of" relationship, for example, Saving Account is a kind of Account, PG student is kind of Student, etc.

The notation for generalization **is a triangle connecting a super class to its subclasses**. The superclass is connected by a line to the top of the triangle. The subclasses are connected by lines to a horizontal bar attached to the base of the triangle. Generalization is a very useful construct for both abstract modeling and implementation. You can see in *Figure 8*, a generalization of Account class.

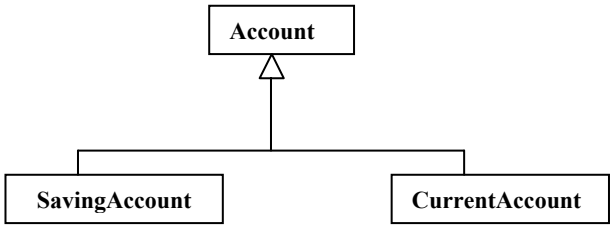

Figure 8: Generalization of account class

Inheritance

Inheritance is taken in the sense of code reuse within the object oriented development. During modeling, we look at the resulting classes, and try to group similar classes together so that code reuse can be enforced. Generalization, specialization, and inheritance have very close association. Generalization is used to refer to the relationship among classes, and inheritance is used for sharing attributes and operations using the generalization relationship. In respect of inheritance, generalization and specialization are two phases of a coin in the sense that if a

subclass is seen from a superclass the subclass is seen as a specialized version of superclass and in, reverse, a superclass looks like general form of subclass.

During inheritance, a subclass may override a superclass feature by defining that feature with the same name. The overriding features (the subclass feature with the same names of superclass features) refines and replaces the overridden feature (the superclass feature).

Now let us look at the diagram given in *Figure 9***.** In this diagram, Circle, Triangle, and Square classes are inherited from Shape class. This is a case of single inheritance because here, one class inherits from only one class.

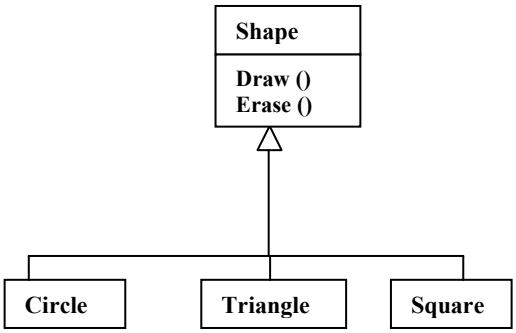

Figure 9: Single inheritance

Multiple inheritance is shown in *Figure 10*. Here, one class is inherited from more than one class.

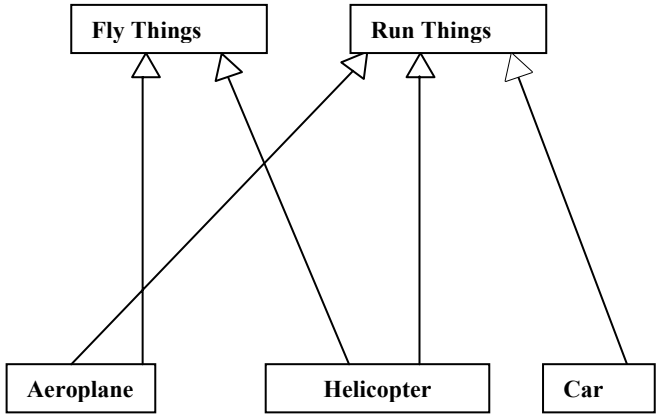

Figure 10: Multiple inheritance

1.5 AN OBJECT MODEL

In object oriented modeling system understanding and on the basis of that identification of classes. Establishing relationship among different classes in the system are the first and foremost activity. Here, we have a simple model of a University System with respect to different levels of courses offered by the University. As you can see in *Figure11*, we have given the basic classes of this system.

This diagram covers different levels of students in the hierarchy. Similarly, for other classes, such as Administration and Faculty, hierarchy level can be drawn to give a broader view of whole system.

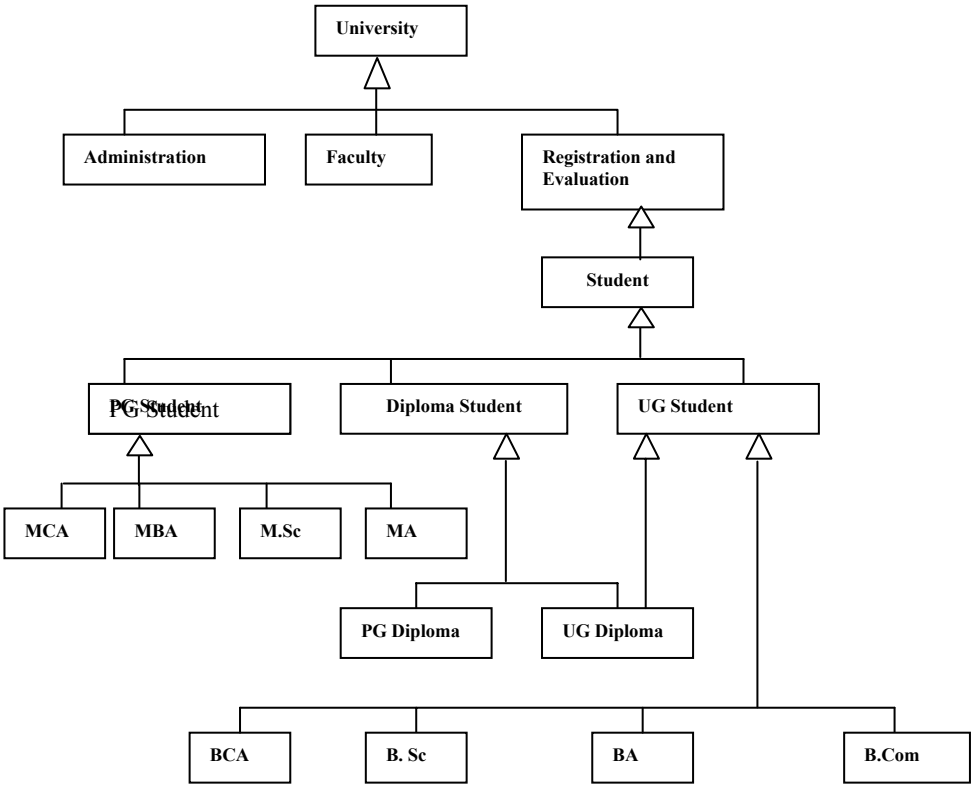

Figure 11: Object model for university system

1.6 BENEFITS OF OBJECT ORIENTED MODELING

There are several advantages and benefits of object oriented modeling. Reuse and emphasis on quality are the major highlights of OOM. OOM provides resistance to change, encapsulation and abstraction, etc. Due to its very nature, all these features add to the systems development:

- Faster development
- Increased Quality
- Easier maintenance
- Reuse of software and designs, frameworks
- Reduced development risks for complex systems integration.

The conceptual structure of object orientation helps in providing an abstraction mechanisms for modeling, which includes:

- Classes
- Objects
- Inheritance
- Association etc.

1.7 INTRODUCTION TO OBJECT ORIENTED ANALYSIS & DESIGN: TOOLS

Unified Modeling Language (UML) is a well accepted language for OOAD. It is used for visualizing, specifying, constructing, and in final documentation. The basic building blocks of UML used for OOAD are things, relationships, and diagrams. Basically, the *Unified Modeling Language* focuses on the concepts of Booch, OMT, and Object Oriented Software Engineering (OOSE). The result of these concepts is a single, common, and widely usable modeling language for users of these and other

methods. The *Unified Modeling Language also* promotes the concept of what can be done with existing methods.

Many modern applications are being developed based on object oriented principles such as classes, methods, and inheritance. To fulfil the needs of such developments, CASE tools offer many benefits for developers building large-scale systems. CASE tools enable us to abstract away from the mess of source code, to a level where design and propose become more clear and easier to understand and modify.

For making and representing these features of the systems to be developed, some tools are used. These tools support UML features and building blocks. Object modeling CASE tools provide support for object oriented modeling notations and methodologies, and they also generate parts of object oriented applications. New versions of many object oriented CASE tools are beginning to address new languages such as Java. Many of these object modeling CASE tools also support relational databases by performing arts of logic, and in some cases, physical database modeling and design, including schema generation and reverse engineering of RDBMS tables, and other elements.

Here, we have tried to give a list collected from many sites and individual searches, for UML modeling tools. Since UML is a fast growing engineering this list may keep on changing all the time.

Action Semantics: This is a group of firms that have responded to the OMG's RFP to define the action semantics for UML.

ArgoUML: This is a domain-oriented design environment that provides cognitive support of object oriented design. ArgoUML provides some of the same automation features of a commercial CASE tool.

ARTiSAN Software: This provides a variety of UML based CASE tools, including a real time modeling tool.
BridgePoint : This provide features of a real time UML modeling tool.

GDPro : this is a full suite of UML and code management tools.

MagicDraw UML: This has Full support for all UML diagrams: MagicDraw RConverter allows you to convert these UML diagrams into MagicDraw: Activity, Class, Collaboration, Component, Deployment, Sequence,State chart, Three-tiered, and Use Case diagrams.

Rational Rose: IBM Rational RequisitePro is a powerful and easy-to-use tool for requirements and use case management.

Visio 2000 Enterprise: It contains a UML suite that can build diagrams within Visio.

Visual Paradigm: Visual Paradigm for the Unified Modeling Language (VP-UML) is a UML CASE suite. The suite of tools is designed for a wide range of users, including Software Engineers, System Analysts, for building large scale software systems reliably through the use of the object oriented approach.

☞ Check Your Progress 3

1) What is inheritance?

……..…

……..….

2) Give an example of multiple inheritance.

……...…

…….

3) Explain the benefit of OOM.

……..…

……..…

……..

1.8 SUMMARY

In this unit we have discussed the basic notions of object orientation, and the need for object oriented modeling. It is the basic characteristic of object orientation which makes it possible to develop systems in such a way that the system is open for reusability. In this unit, concepts of abstraction, encapsulation, polymorphism and sharing of structure and behaviour are discussed.

Further, in this unit, we have discussed notions of class and object. We saw that how inheritance, generalization/specialization, and associations are represented. In this unit, a hierarchy of classes representing different levels of students in a University system is representated, reuse and quality are mentioned as benefits of OOM, and in the last section, some tools, which support UML designs are mentioned.

1.9 SOLUTIONS/ANSWERS

Check Your Progress 1

1) Object oriented modeling is a approach through which modeling of the systems are done by visualizing the system based on the real world concepts. Object oriented modeling is language independent and generally used for complex system development.

2) Steps involve in OOM are:

   System Analysis

   System Design

   Object Design and

   Final implementation.

3) Structured approach of problem solving is based on the concept of decomposition of system in to subsystem. In this approach of system development readjustment of some new changes in the system is very difficult. On the other hand in object oriented approach holistic view of application domain is considered and related object are identified. Further classification of objects are done. Object oriented approach give space for further enhancement of the system without too much increase in systems complexity.

Check Your Progress 2

1) Abstraction in object orientation is a concept which provide opportunity to express essential properties of the object without providing much details of implementation of these properties.

2) Association is used for establishing relationships between classes. Association describe links between (among) classes. For example, if a professor works in a university then it can be represented as

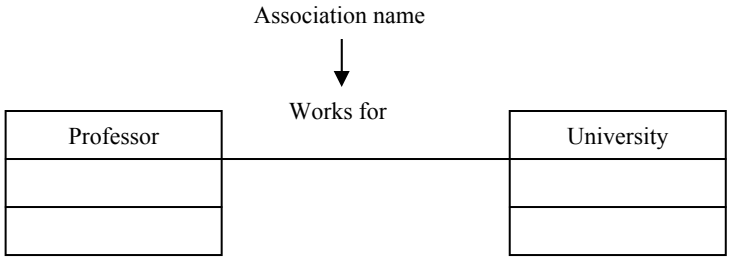

Figure 12: Association of professor and university

3) Multiplicity in an association indicate the number of objects participate in a relationship. For example in the association given in *Figure 13* you can see that one player can play for one team at a time so here multiplicity is 1.

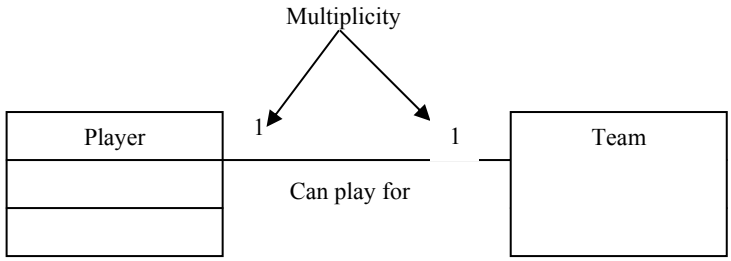

Figure 13: Multiplicity in association

Check Your Progress 3

1) Inheritance is an object orientation concept which allow reusability of design/code. Basic meaning of inheritance is that if one class is already defined than another class which also passes the property of existing class can be defined and inherit the property of existing class. For example, if a class named student is defined and another class for Post Graduate students is to be defined then PG Student class can inherit student class.

2) One example of multiple inheritance is a committee for students affair which consist of faculty and administrative staff member.

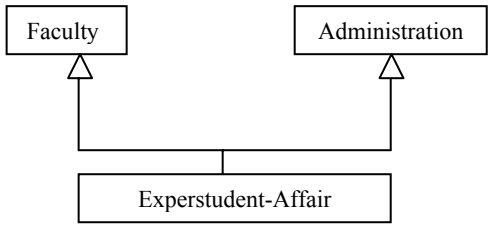

Figure 14: Multiple inheritance

3) Major benefits of object oriented modeling is that development of the system become fast, quality of the system is increase. It give freedom of use of existing design and code. Help in development of complex system with less risk due to the basic properties of object orientation which include class, objects and inheritance.

UNIT 2 OBJECT ORIENTED ANALYSIS

Structure **Page Nos.**

2.0 INTRODUCTION

Object oriented analysis (OOA) is concerned with developing software engineering requirements and specifications that expressed as a system's object model composed of a population of interacting objects. The concept of abstraction in object oriented analysis (OOA) is important. Abstraction may be defined as: the characteristics of an object which make it unique and reflect an important concept. Analysis is a broad term, best qualified, as in *requirements analysis* (an investigation of the requirements) or *object oriented analysis* (an investigation of the objects of the problem domain).

OOA views the world as objects with data structures and behaviors and events that trigger operations, for object behavior changes. The idea is to see system as a population of interacting objects, each of which is an atomic bundle of data and functionality, (the foundation of object technology) and provides an attractive alternative for the development of complex systems.

The problem statement is important for any analysis. It is a general description of the user's difficulties, and desires. The purpose of problem statement is to identify the general domain in which you will be working.

In this Unit you will study various analysis techniques: object modeling, dynamic modeling and functional modeling. You will also learn how add operations in system and how to do refining of the analysis model.

2.1 OBJECTIVES

After going through this unit, you should be able to:

- define the concepts of the objects in the system;
- express required system behaviour in terms of business objects in the system, and actions that the user can perform on them;
- understand how to define and analyze the problem statement;
- explain the purpose of object modeling;
- explain dynamic modeling;
- describe the event flow between objects and how to draw state diagrams of real world problems;
- explain and understand the importance of functional model;
- explain how operations can be added in various analysis techniques, and
- explain the importance of iterating or refining the analysis model.

2.2 OBJECT ORIENTED ANALYSIS

In this section we will discuss basics of object oriented analysis with the help of object oriented features.

Analysis

Analysis is not a solution of the problem. Let us see what actually object oriented (OO) analysis. Analysis emphasizes an ***investigation*** of the **problem** and **requirements,** for example, if a new online trading system is desired by a trader, there may be questions such as, how will it be used? What are its functions?

If you start a project the first question about that project' will be how do we get started? Where do we begin?

The starting point for object oriented analysis is to identify candidate objects and their relationships.

The first stage can be a simple brain-storming of the possible objects. Then one method is to go through all the nouns in any documentation about the world you are analysing, and considering these as candidate objects of the system. Also, you can use the alternative technologies for developing software engineering requirements and specifications that include functional decomposition, essential systems analysis, and structured analysis.

The Process of Development

The approach to development can be an iterative one. It involves repeated refinement of the object model. The process needs to be controlled by an appropriate project management process, involving reviews and check pointing at the levels of:

- Analysis
- Design
- Implementation
- Post Implementation Review

Object Orientation and Analysis

An ***Object*** is something that exists within the problem domain that can be identified by data and/or behaviour. An example of an object is a car. The data of a car could be the wheel, brake, seat, etc. The behaviour of the car would be to drive on roads, its speed, etc.

Object oriented analysis is the concept which actually forces you to think in terms of the application domain when its behaviour and data known to you.

In OOA the primary focus on identifying objects from the application domain, then fitting procedures around them.

For example, in the case of the flight information system, the objects would include *Plane, Flight,* and *Pilot,* etc.

The object model has many aspects, which are associated with OO concepts.
Now we will discuss the following principle of OO.

**Abstraction, Encapsulation, Identity, Modularity, Hierarchy,
Typing, Concurrency, and Persistence**

Abstraction

You understand the term object. Now, let us see how problems are seen as objects, and find their associated data and behaviour. You will notice that an object is a real life entity, or abstraction.

In our daily life we deal with complexity by abstracting details away.

Let us see this with an example of:

Driving a car does not require knowledge of internal combustion engine. It is sufficient to think of a car as simple transport.

In simple term, abstraction means to focus on the essential, inherent aspects of an entity, ignoring its accidental properties. Abstraction is a normal process that we do everyday. When you view the world, you can not possibly take in every minute detail, so you concentrate on the aspects that are important.

An abstraction is a simplified description of a system that captures the essential elements of that system (from the perspective of the needs of the modeler), while suppressing all other elements.

Encapsulation

This separates the interface of an abstraction from its implementation. Taking the above example of a car, we can now categorize as:

Abstraction	Car
Interface	Steering, pedals, controls
Implementation	Generally, you don't know

Encapsulation also means **data hiding**, which consists of separating the external aspects of an object, which are accessible to other objects.

Now let us take a example of the Stack.

A Stack abstraction provides methods like push (), pop (), isEmpty(), isFull(). The Stack can be implemented as a singly linked list, or a doubly linked list, or an array, or a binary search tree. This feature is called encapsulation. It hides the details of the implementation of an object.

The Benefits of Encapsulation

To hide the details of a class, you can declare that data or implementation in its private part so that any other class clients will not be able to know about. It will have the ability to change the representation of an abstraction (data structures, algorithms) without disturbing any of its clients. The major benefit of encapsulation is that you.

By Encapsulation

You can delay the resolution of the details until after the design.
Keep your code modular.

Object Identity

Object identity is a property of an object that distinguishes the objects from all other objects in the applications. With object identity, objects can contain, or refer to other objects. You can create an object identity in three ways:

1) You can refer as memory address in programming languages.

2) Assign identifier keys in the database.

3) By user-specified names, used for both programming and database.

In a complete object oriented system each object is given an identity that will be permanently associated with the object irrespective of the object's structural or state transitions. The identity of an object is also independent of the location, or address of the object. Object identity provides the most natural modeling primitive to allow the "same object to be a sub-object of multiple parent objects".

Modularity

Modularity is closely tied to encapsulation; you may think of modularity as a way of mapping encapsulated abstractions into real, physical modules. It is a property of a system that has been decomposed into cohesive and loosely coupled modules.

Cohesion and coupling gives two goals for defining modules. You should make a module cohesive (shared data structures, similar classes) with an interface that allows for minimal inter-module coupling.

It is important to remember that the decisions concerning modularity are more physical issues, whereas the encapsulation of abstractions is logical issues of design.

Hierarchy

This is ranking or ordering of abstraction.
Hierarchy is decided by using the principle of 'divide and conquer'. You can describe complex objects in terms of simpler objects. This is the property of object oriented analysis (OOA) which enables you to *reuse the code.*

You can place an existing class directly inside a new class. The new class can be made up of any number and type of other objects, in any combination that is needed to achieve the desired functionality. This concept is called *composition* (or more generally, *aggregation*). Composition is often referred to as *a "has-a"* relationship *or "part-of"* relationship, for example, in "automobile has an engine" or "engine is part of the automobile."

Typing

This enforces object class such that objects of different classes may not be interchanged. In other words, class can be visualized as a type. Remember that there are two kinds of typing. This typing does not mean the way you usually type the letters.

Strong Typing: When any operation upon an object (which is defined) can be checked at compile time, i.e., type is confirmed forcefully.

Weak Typing: Here, operations on any object can be performed, and you can send any message to any class. Type confirmation is not essential, but in these type of language more errors at execution time may occur.

Concurrency

The fundamental concept in computer programming is the idea of handling more than one task at a time. Many programming problems require that the program be able to:

1) Stop what it's doing

2) Currently deal with some other problem, and

3) Return to the main process. There is a large class of problems in which you have to partition the problem into separately running pieces so that the whole program can be more responsive. Within a program, these separately running pieces are called threads, and the general concept is called *multithreading.*

Currently, if you have more than one thread running that is expecting to access the same resource then there is a problem. To avoid this problem, a thread locks a resource, completes its task, and then releases the lock so that someone else can use the resource. It can lock the memory of any object so that only one thread can use it at a time. It is important to handle concurrent running programs/threads properly.

Persistance

When you create an object, it exists for as long as you need it, but under no circumstances do object exist when the program terminates. While this makes sense at first, there are situations in which it would be incredibly useful if an object could exist

and hold its information even while the program is not running. When, next time you start the program, the object would be there and it would have the same information it had the previous time the program was running. Of course, you can get a similar effect by writing the information to a file or to a database, but in the spirit of making everything an object it would be quite convenient to be able to declare an object persistent and have all the details taken care of for you.

2.3 PROBLEM STATEMENT: AN EXAMPLE

You must understand here that you are looking for a statement of needs, not a proposal for a solution. OOA specifies the structure and the behaviour of the object, that comprise the requirements of that specific object. Different types of models are required to specify the requirements of the objects. These object models contain the definition of objects in the system, which includes: the object name, the object attributes, and the objects relationships to other objects.

As you know, an object is a representation of a real-life entity, or an abstraction. For example, objects in a flight reservation system might include: an airplane, an airline flight, an icon on a screen, or even a full screen with which a travel agent interacts. The behaviour, or state model describes the behavior of the objects in terms of the states of the object and the transitions allowed between objects, and the events that cause objects to change states.

These models can be created and maintained using CASE tools that support the representation of objects and object behaviour.

You can say that the problem statement is a general description of the user's difficulties, desires, and its purpose is to identify the general domain in which you will be working, and give some idea of why you are doing OOA.

It is important for an analyst to separate the true requirements from design and implementation decisions.

Problem statement should not be incomplete, ambiguous, and inconsistent. Try to state the requirements precisely, and to the point. Do not make it cumbersome.

Some requirements seem reasonable but do not work. These kind of requirements should be identified. The purpose of the subsequent analysis is to fully understand the problem and its implications, and to bring out the true intent of the Client.

☞ Check Your Progress 1

1) Give two benefits of Reuse of Code.

……

…..……...

.....

2) Give an example of enforcement in Typing.

……

…..……...

……

3) What are the benefits of OOA technology?

……

…..……..

……

4) Briefly explain what is to be done while defining the problem statement.

……

…..……...…

……

You are already familiar with structured analysis. Now, this is the appropriate time to have a comparison of OOA and Structured analysis.

2.4 DIFFERENCES BETWEEN STRUCTURED ANALYSIS AND OBJECT ORIENTED ANALYSIS

Object oriented analysis design (OOAD) is basically a bottom up approach which supports viewing the system as a set of components (objects) that can be logically kept together to form the system.

Advantages and Disadvantages of Object Oriented Analysis and Design

Advantages:

The OO approach inherently makes each object a stand alone component that can be reused not only within a specific stat problem domain, but also is completely different problem domains, having the requirement of similar objects.

The other main advantage of object oriented (OO) is the focus on data relationships. You cannot develop a successful system where data relationships are not well understood. An OO model provides all of the insight of an ER diagram and contains additional information related to the methods to be performed on the data. We will have more detailed discussion on this aspects in Block 3 of this Course.

Disadvantages:

You know that OO methods only build functional models within the objects. There is no place in the methodology to build a complete functional model. While this is not a problem for some applications (e.g., building a software toolset), but for large systems, it can lead to missed requirements. You will see in Unit 3 of this course. “Use cases” addresses this problem, but since all use cases cannot be developed, it is still possible to miss requirements until late in the development cycle.

Another disadvantage of the object oriented analysis design (OOAD) is in system modeling for performance and sizing. The object oriented (OO) models do not easily describe the communications between objects. Indeed, a basic concept of object oriented (OO) is that the object need not know who is invoking it. While this leads to a flexible design, performance modeling cannot be handled easily.

The object oriented (OO) analysis design itself does not provide support for identifying which objects will generate an optimal system design. Specifically, there is no single diagram that shows all of the interfaces between objects. You will study object oriented analysis design (OOAD) diagrams in Unit 3 of this course. As you know, coupling is a major factor in system complexity, not having this information makes architecture component selection a hit or miss proposition.

Advantages and Disadvantages of Structured Analysis

With experience, you will come to know that most customers understand structured methods better than object oriented (OO) methods. Since one of the main reasons of modeling a system is for communication with customers and users, there is an advantage in providing structured models for information exchange with user groups or customers.

In fact, specifications are typically in the form of a simple English language statement of Work and Requirement Specification. Therefore, the system to be built, must be understood in terms of requirements (functions the system must perform), that is why this naturally leads to a structured analysis, at least at the top level. Specifically structured methods (functional decomposition) provide a natural vehicle for discussing, modeling, and deriving the requirements of the system.

The disadvantage with structured methods is that they do not readily support the use of reusable modules. The top down process works well for new development, but does not provide the mechanisms for "designing in" the use of existing components. The top down process of functional decomposition does not lead to a set of requirements which map well to existing components.

When the requirements do not map cleanly, you have two choices: either you do not use the existing components, or force fit the requirements to the existing components and "somehow" deal with the requirements which are only partially covered by the existing components, which does not lead to a good successful system. Now, we will discuss how actually object oriented analysis (OOA) system is performed.

2.5 ANALYSIS TECHNIQUES

In this section we will see how classes re identified, and their applications with the help of basic modeling techniques.

2.5.1 Object Modeling

Object modelling is very important for any object oriented development, object modeling shows the static data structure of the real world system. Basically, object modeling means identifying objects and classes of a system, or you can say that it describes real world object classes and their relationships to each other. You can develop an object model by getting information for the object model from the problem statement, expert knowledge of the application domain, and general knowledge of the real world. Object model diagrams are used to make easy and useful communications between computer professionals and application domain experts.

To develop an object model first identify the classes and their associations as they affect the overall problem structure and approach. Then prepare a data dictionary.

i) Identify associations between objects.
ii) Identify attributes of objects and links.
iii) Organise and simplify object classes using inheritances.

Then you verify that access paths exist for likely queries

iv) Iterate and refine the model, and
v) Group classes into modules.

Identifying Object Classes

You should always be careful while identifying relevant object classes from the application domain. Objects include physical entities in a system like buildings, employees, department, etc. Classes must make sense in the application domain. At this stage you should avoid computer implementation constructs, such as linked lists and subroutines.

Some classes are implicit in the application domain, so you need to find out by understanding the problem well.

Action-object matrix: A matrix showing how update actions affect objects. It may be considered to be part of the user object model, as it summarizes user object action definitions in a tabular view.

Process of this whole activity is like:

Check for multiple models

- Identify objects
- Create user object model diagram
- Define user object attributes
- Define user object actions
- Create action-object matrix
- Check for dynamic behavior
- Review glossary.

Some notations for user object model:

Notation Meaning

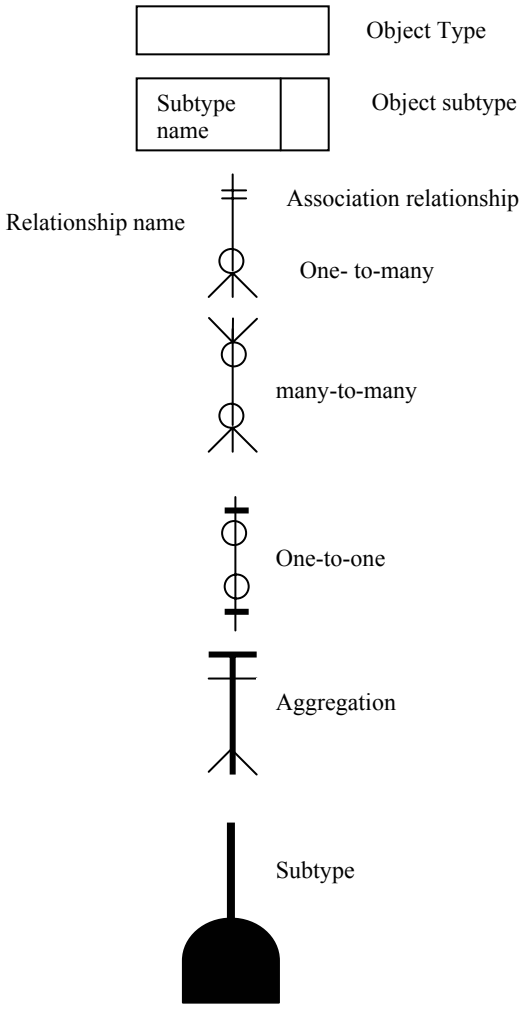

Figure 1: User Object Model Notations

Let us see the example in which we will discuss how concepts are implemented:

To illustrate the object modeling technique to you by taking the very common example of the tasks associated with using a telephone set:

Answer the “phone	
Hear the “phone ringing.	[Ringer]
Identify the ringing “phone.	[Receiver Unit]
Pick up the handset.	[Handset]
Talk to the caller.	[Other party]
Replace the handset.	[Handset]
Make a ‘phone call.	
Pick up the handset.	[Handset]
Listen for the dialing tone.	[Exchange]

'Dial' the number. [Dial]
Listen for the ringing tone. [Receiver Unit]
Talk to the person answering. [Other party]
Replace the handset. [Handset]

On the basis of above information, the **objects/actions** that may be identified from this are:

Receiver Unit
Identify
Ringer
Hear
Handset
Pick up
Replace
Other party
Talk with
Exchange
Listen to
Dial
Enter number

The attributes of these identified objects are identified as:

Receiver Unit
Identity (phone number)
Status (ringing, engaged)
Ringer
Ringing (true, false)
Handset
Hook (on, off)
Other party
Status (caller, answered)
Exchange
Tone (dead, dialing, other)
Dial
Status (empty, number entered)

You can see the User Object Model diagram produced from this given in *Figure 2.*

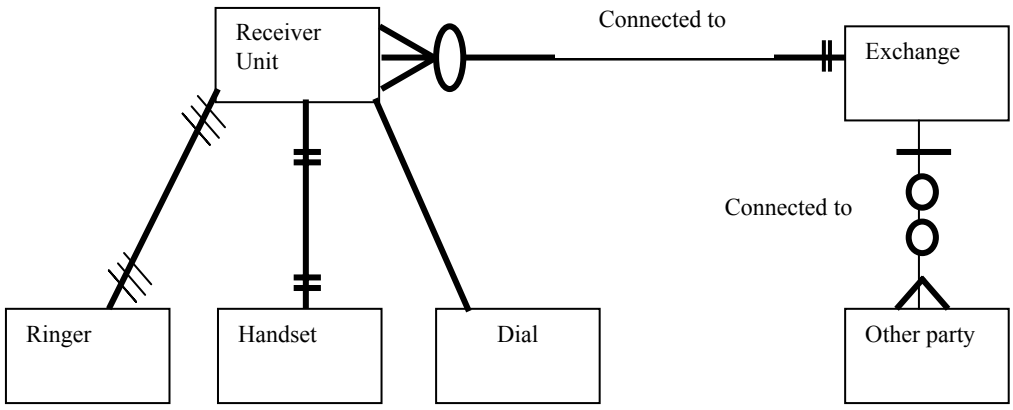

Figure 2: User object Model for Telephone set

2.5.2 Dynamic Modeling

You know that computer systems are built from objects which respond to events. External events arrive at the boundary of the system; you understand the concept of events.

For example, whenever you click with your mouse on the desktop or the canvas area some action is triggered (you get some response back).

In Dynamic modeling you examine "**a way of describing how an individual object responds to events, either internal events triggered by other objects or external events triggered by the outside world**".

Dynamic modeling is elaborated further by adding the concept of time: new attributes are computed as a function of attribute changes over time.

Before you define the dynamic behaviour of user objects. You should know the following:

The Inputs are:

User object model which comprises objects, attributes, actions and relationships.
Task model: from which significant object states may be identified.

Products

Dynamic model: A model of the dynamic behaviour of a user object. It defines significant states of the user object, the way that actions depend on the state, and affect the state.

The dynamic model consists of a dynamic model diagram, showing states and transitions and supplementary notes, specifying states and actions in more detail.

Process of Dynamic modeling:

- Analyse applicability of actions
- Identify object states
- Draw dynamic model diagram
- Express each state in terms of object attributes
- Validate dynamic model
- Concepts.

Dynamic modeling: state diagrams

We will study this in detail in Block 3 of this course.

Generally, a state diagrams allows you to further explore the operations and attributes that need to be defined for an object. They consist of sets of states which an object is in, and events which take the object from one state to another.

State: An object may have one or more states−stable points in its life, expressed by the object's attributes and relationships.

Event/action: Something that happens to an object. Atomic, in that it either has happened or it hasn't. An event causes an action.

Transition:

A jump between states, labeled with the corresponding action.

Notations for state diagram are shown below in *Figure 3*:

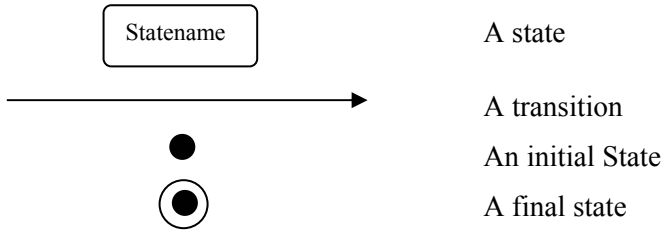

Figure 3: Notation for State Diagram

A simple example using these notation is shown below in Figure 4.

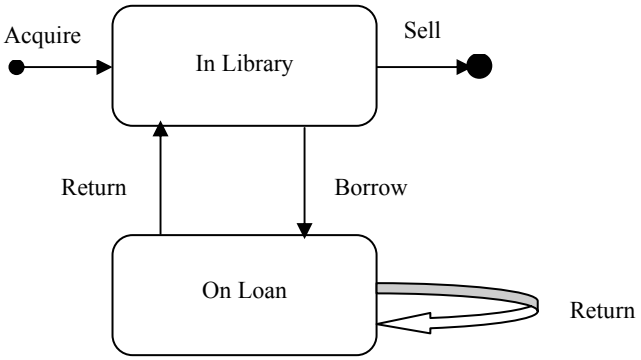

Figure 4: A simple State Diagram

States may be seen as composite states: a collection of states treated as one state at a higher level on analysis. In the example above, the state "In Library" is actually composed of a sequence of sub-states which the library staff, if not borrowers would need to know about.

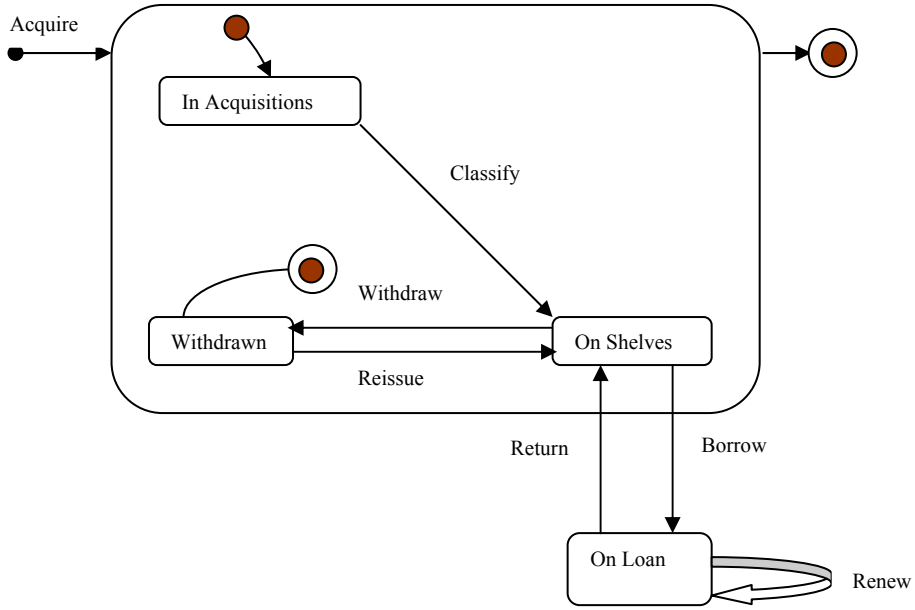

Figure 5: Composite States

Basically any system which you can reasonably model on a computer jumps from state to state.

The level of state decomposition must be decided by judgement. A too fine grained model is inappropriate; for example, modeling all the possible ages of an employee as individual states. Also, a gross decomposition is useless; for example, modeling an employee as employed or not.

You can ask whether all objects of world have behaviour which can be modeled.

Of course, not all objects have behavior worth modeling. Consider our example from the field of object oriented nursery rhymes, the sad story of Humpty Dumpty. Clearly,

you would not like to model such trivial examples.

Basically you need to construct a state transition diagram for every object with significant behaviour. You need not construct one for anything with trivial behaviour.

The reason for doing this is to identify further operations and attributes, to check the logical consistency of the object and to more clearly specify its behaviour.

All the best notations can be used to describe the process they facilitate.

Now we have the basic ideas of object models and dynamic models, our approach to analysis and design (so far) can be summarised as:

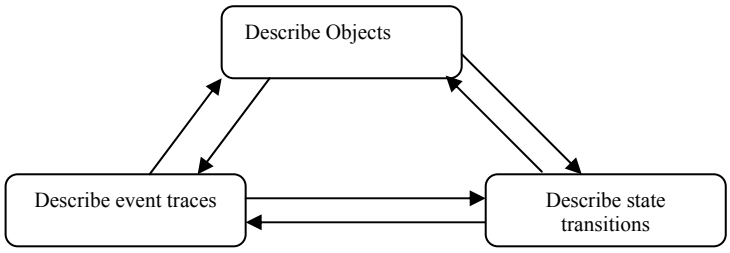

Figure 6: Objects and State transitions

Note: You should also match the events between state diagrams to verify consistency. Till so far, you have been introduced to object modeling, and dynamic modeling. Now, we will see what Function modeling is:

2.5.3 Functional Modeling

You know that Data flow modeling is a common technique used for the analysis of a problem in software engineering. It is exceptionally useful for analyzing and describing systems where there is a large amount of calculation involved.

Data flow models consist of a number of processes which exchange information. A process transforms information it receives, and passes the transformed information on to other processes or to objects in the system. Data flow models can be used to uncover new operations and new attributes for the object model. Sometimes, new objects can be discovered too.

Basically you can state that the functional model shows how values are computed. It describes the decisions or object structure without the regard for sequencing.
It gives you dependency between the various data and the functions that relate them, giving the flow of data.

Each process needs to be implemented as an operation in one or more of the objects. Each data item arising from an object must have a corresponding attribute, or set of attributes in the source object. The data flow diagram (DFD) corresponds to activities or actions in the state diagrams of the classes.

That is why it is suggested to construct the functional model after the object and dynamic models.

Now, let us discuss about the creation of the DFD and the functional model.

The approach to data flow diagramming should be as follows:

- Create a data flow diagram for each of the major outputs of the system
- Work back from the outputs to the inputs to construct the diagram
- Add new objects where necessary to the object model as you discover the need for them in the data flow modeling add new operations and attributes to the object model as you discover the need for them in the data flow modeling.

One thing you have to remember is that the data flow diagram is not used as a basis for devising the structure of the system.

Steps in constructing a Functional Model

- Identify input and output values
- Build data flow diagrams showing functional dependencies
- Describe functions
- Identify constraints
- Specify optimisation criteria.

Identifying Input and Output Values

First, identify what data is going to be used as input to the system, and what will be the output from the system. Input and output values are parameters of events between the system and the outside world. You must note that Input events that only affect the flow of control, such as cancel, terminate, or continue. They do not supply input values. For example, supplier code, name, product description, rate per unit, etc. are the inputs to a sales system.

Build data Flow diagrams showing functional dependencies

Data flow diagrams are useful for showing the functional dependencies. In data flow diagrams processes are drawn as bubbles, each bubble containing with the name of the process inside. Arrowhead lines are used to connect processes to each other, and to objects in the system. The lines are label with the information that is being passed. Objects are drawn as rectangular boxes, just as in the object model, but usually with just the name of these objects and not the attributes and operations.

i)

Sales
Quantity delivered
Calculate quantity delivered to customer

iii)

Product
name price vat rate

ii)

Customer
Product line discount quantity discount
Calculate product discount calculate quantity discount

iv)

Invoice
Calculate line item discount Calculate total Deduct quantity discount Print

Figure 7: Some objects in figure 7: a simple DFD is given for sales system

Let us look at a simple example:

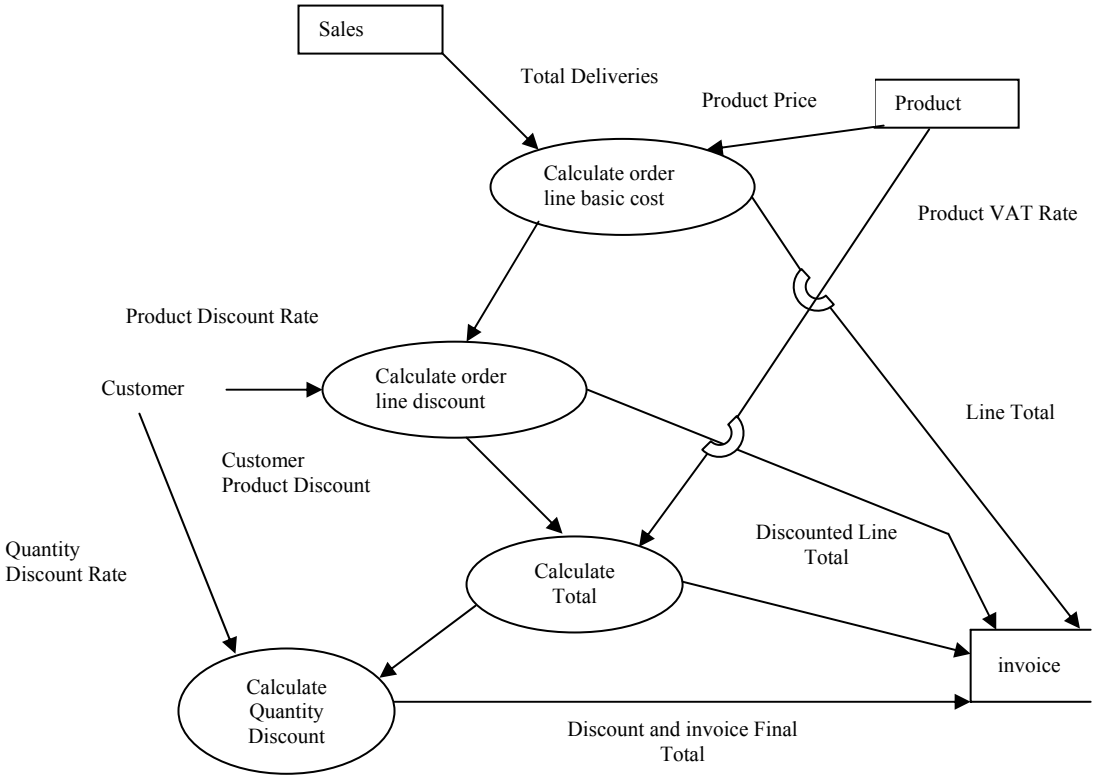

Figure 8: DFD for Sales System

The next stage is to devise operations and attributes to support the calculations.

Describe functions

After you have roughly designed the data flow diagram, you can write a description of each function, and you can describe the function in any form such as, mathematical equations, pseudo code, decision tables, or some other appropriate form. You need not know the implementation of the function, but what it does. The description can be declarative or procedural.

A declarative description specifies the relationship between the input and output values, and relationship among the output values.

A procedural description specifies a function by giving an algorithm to compute it. The purpose of the algorithm is only to specify what the function does.

Identifying Constraints Between Objects

In a system, there are some boundaries to work on. Boundaries or Constraints are the functional dependencies between objects which are not related by an input-output dependency. You have to find out the constraints between the objects. Constraint has different behavior at different times. It can be on two objects at the same time, between instances of the same object at different times (an invariant), or between instances of different objects at different times.

You can define constraints as Preconditions (input values) and PostConditions (output values). Preconditions on functions are constraints that the input values must satisfy. Post conditions are constraints that the output values must satisfy to hold. State the conditions under which constraints hold.

Specifying Optimisation Criteria

Specify values to be maximized, minimized, or optimized. You can understand it as the way you normalize the data in the database. For example, you should minimize the time an account is locked for concurrency reasons. If locks are needed, then it is extremely important to minimize the time that an entire bank is locked for concurrency reasons.

☞ Check Your Progress 2

1) Explain how you can define an object model of a system.

……

….……...…

……...

2) Show the basic dynamic model of telephone.

……

….……

2.6 ADDING OPERATIONS

Whenever you look at the operations in OOPs you find queries about attributes or associations in the object model (such as student.name), to events in the dynamic model (such as ok, cancel), and to functions in the functional model (such as update, save).

You can see operations from the object model. It includes reading and writing attribute values and association links. In the object model operations are presented with an attribute. During analysis nothing is private, you assume that all attributes are accessible.

Accessing from one object to another through the object model can be referred to as "pseudo-attribute" like Account.Bank, where account and bank are two separate objects of their respective classes.

Operations from events

During analysis, events which are sent to the target objects. An operation on the object are presented as labels on transitions and should not be explicitly listed in the object model.

Events can be expressed as explicit methods.

You can also implement events by including event handler as part of the system substrate.

Operations from State Action and Activities

You must see that State actions and activities are actually functions, which can be defined as the operations on the object model.

Operations from Functions

As you know, function are actually operations on object. These functions should be simple and summarized on the object model. Organise the functions into operations on objects. For example, the select operations are really path traversals in the object model. The operations like withdrawal-money, verify-password are the operations on class Account of Bank Management system.

You can write as:
account: withdraw (code, amount)->status
account: deposit (code, amount)->status.

☞ Check Your Progress 3

1) Describe how you can simplify Operations.

……

……

……

2) Give an example of operations from State Actions and Activities.

……

……

……

3) Give an example of Operations from functions of a bank.

……

……

……

4) How do you see the final model after iterative analysis?

……

……

……

5) Why is iterative analysis of any problem needed?

……

……

……

Iteration of analysis is important. Let us see how OOA apply iteration

2.7 ANALYSIS ITERATION

To understand any problem properly you have to repeat the task which implies that analysis requires repetition. First, just get the overview of the problem, make a rough draft, and then, iterate the analysis according to your understanding. Finally, you should verify the final analysis with the client or application domain experts. During the iteration processes refining analysis and restating of requirement takes place.

2.7.1 Refining the Ratio Analysis

Basically, refinement leads to purity. So to get a cleaner, more understandable and coherent design, you need to iterate the analysis process.

- Reexamine the model to remove inconsistencies, and imbalances with and across the model.
- Remove misfit, and wrong concepts from the model.
- Refining sometimes involves restructuring the model**.**
- Define constraints in the system in a better way.
- You should keep track of generalizations factored on the wrong attributes.
- Include exceptions in the model, many special cases, lack of expected symmetry, and an object with two or more sets of unrelated attributes, or operations.
- You should remove extra objects or associations from the model. Also, take care to remove redundancy of objects and their extra attributes.

2.7.2 Restating the Requirements

To have clarity of the analytical model of the system you should state the requirements specific performance constraints with the optimization criteria in one document verify in the other document. You can state the method of a solution.

Verify the final model with the client. The requirement analysis should be confirmed and clearly understood by the client**.**

The impractical, or incorrect, or hypothetical objects that do not exist in the real world should be removed from the proposed system.

You must note that the analysis model is the effective means of communication with application experts, not computer experts. In summary, the final model serves as the basis for system architecture, design, and implementation.

2.8 SUMMARY

In this Unit you have learned that the goal of analysis is to understand the problem and the application domain, so that you come out with a well cohesive design. There is no end or border line between the analysis and the design phase in software engineering. There are three objectives of the analysis model:

Object oriented analysis (OOA) describes what the customer requires, it establishes as basis for the creation of a software design, and it defines a set of requirements that can be validated.

You also remember that the requirement analysis should be designed in such a way that it should tell you what to be done, not how it is implemented.

The object model is the principal output of an analysis and design process.

Dynamic modeling is elaborated further by adding the concept of time: new attributes are computed as a function of attribute changes over time. In this unit, after defining the **scenario** of **typical** and **exceptional sessions**, identify events followed by the building of state diagrams for each active object showing the patterns of events it

receives and sends, together with actions that it performs. You should also match the events between state diagrams to verify consistency.

You have learned that a functional model shows how values are computed; it describes the decisions, or object structure without regard for sequencing.

It gives you dependency between the various data and the functions that relate them, giving the flow of data. Here you construct the data flow diagram which interacts with internal objects and serves as, data stores between iterations. You also specify the required constraints, and the optimisation criteria.

You have seen that refining and restating will give you more clarity of the analytical model of the system.

2.9 SOLUTIONS / ANSWERS

Check Your Progress 1

1) **Reusing the implementation**. Place an existing class directly inside a new class. The new class can be made up of any number and type of other objects, in any combination that is needed to achieve the desired functionality. This concept is called *composition* (or more generally, *aggregation*). Composition is often referred to as a "has-a" relationshp or "part-of" relationship, as in "automobile has an engine" or "engine is part of the automobile."

   Reusing the interface. Take an existing class and make modifications or additions to achieve desired functionality. This concept is called *inheritance*. The original class is called the Base class or Parent class and the modified class is called the *Derived* or *Inherited* or *Sub* or *Child* class.

2) You can understand the concept of enforcement as it make sure objects of different classes may not be interchanged as follows:

   Example: Vegetable v; Fruit f; Mango m;

   This implies 'v' is variables of class Vegetable, 'f' of class Fruit and 'm' of class Mango. Typing ensures that value of 'f' cannot be assigned to 'v'. However, if Mango extends Fruits, then 'f' can be assigned a value of 'm'. The variable of a sub class can be assigned to variable of super class, but not the other way around, and 'm' cannot be assigned a value of 'f'.

3) Using the OOA technology can yield many benefits, such as:

   i) Reusability of code
   ii) Productivity is gained gains through direct mapping
   iii) Maintainability, through this simplified mapping to the real world is possible.

4) To define the Problem Statement of a system, define what is to be done, and how you are going to implement that. Your statement should include the mandatory features, as well the optional features of the system to be developed.

   You should include:
   What is the problem and its scope,
   What is needed
   Application context
   Assumptions
   Performance needs.

Check Your Progress 2

1) A list of terms which will be used by end users to describe the state and behaviour of objects in the system.

Different user classes may require different mental models of the objects in the system. This includes:
What type of objects there are (user objects).
What information the user can know about an object of a particular type (user object attributes).
How the objects may be related to other objects (relationships).
Object types with 'subtypes' which have additional specialised actions or attributes, i.e., User object, Container objects, User object action, User object subtype.
A model of the business objects which end users believe interact with in a GUI system.

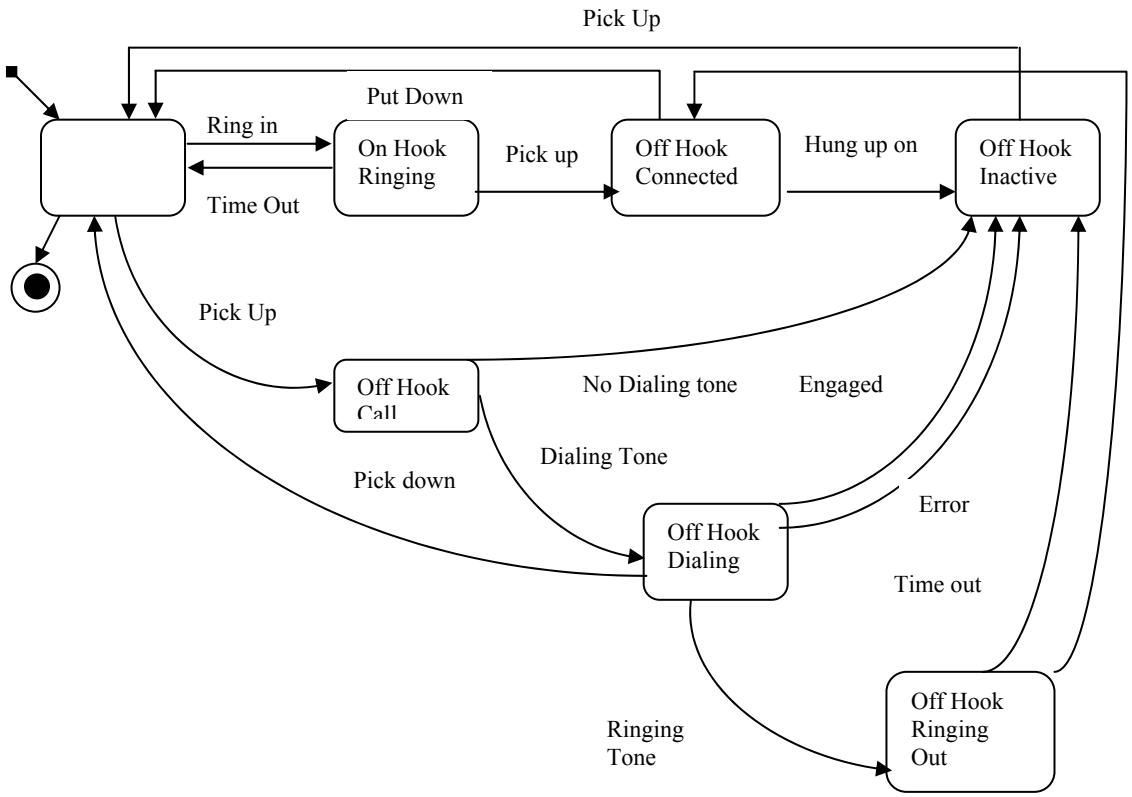

Figure 9: Dynamic Model of Telephone

Check Your Progress 3

1) To simplify the operation, one should use inheritance, where possible, to reduce the number of distinct operations. Introduce new superclasses as needed to simplify the operations, but they should not be forced or unnatural.
Locate each operation at the correct level in the class hierarchy.

2) For example, in the bank the activity, *verify account code* and *verify password*

3) For the creation of a saving account, you may write:
bank::: create –savings-account (customer)->account.
For the operation of checking account, you can write:
bank:: :create-checking-account

4) The final model serves as the basis for system architecture, design, and implementation.

5) The iterative analysis is required to get cleaner, more understandable, and coherent design.

UNIT 3 USING UML

Structure **Page Nos.**

3.0 INTRODUCTION

One of the major issues in software development today is quality. Software needs to be properly documented and implemented. The notion of software architecture was introduced for dealing with software quality. For successful project implementation the three essential components are: process, tools and notations. The notation serves three roles:

- as the language for communication,
- provide semantics to capture strategic and tactical decisions,
- to offer a form that is concrete enough to reason and manipulate

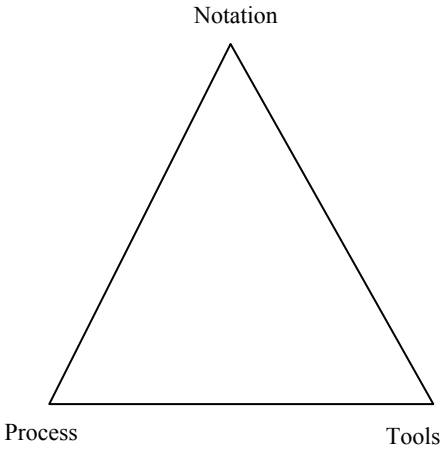

Figure 1: Components of project implementation

Architectural description languages (ACLs) have been developed for architectural description in analysis and design process. Important architectural description languages are Rapide, Unicorn, Asesop, Wright, ACME, ADML and UML. Currently, Universal Modeling Language (UML) is a *de facto* standard for design and description of object oriented systems, and includes many of the artifacts needed for architectural description, such as like processes, nodes, views, etc.

3.1 OBJECTIVES

After going through this unit, you should be able to:

- trace the development of UML;
- identify and describe the notations for object modeling using UML;
- describe various structural and behavioral diagrams;
- list the characteristics of various static and dynamic diagrams, and
- understand the significance of different components of UML diagrams.

In this Unit we will discuss object modeling notations, structured diagrams and behavioral diagrams of systems.

3.2 UML: INTRODUCTION

The Unified Modeling Language (UML) is used to express the construct and the relationships of complex systems. It was created in response to a request for proposal (RFP) from the Object Management Group (OMG). Earlier in the 1990s, different methodologies along with their own set of notations were introduced in the market. The three prime methods were OMT (Rumbaugh), Booch and OOSE (Jacobson). OMT was strong in analysis, Booch was strong in design, and Jacobson was strong in behavioral analysis. UML represents unification of the Booch, OMT and OOSE notations, as well as are the key points from other methodologies. The major contributors in this development shown in *Figure 2*.

UML is an attempt to standardize the artifacts of analysis and design consisting of semantic models, syntactic notations and diagrams. The first draft (version 0.8) was introduced in October 1995. The next two versions, 0.9 in July 1996 and 0.91 in October 1996 were presented after taking input from Jacobson. Version 1.0 was presented to Object Management Group in September 1997. In November 1997, UML was adopted as standard modeling language by OMG. The current version while writing this material is UML 2.0.

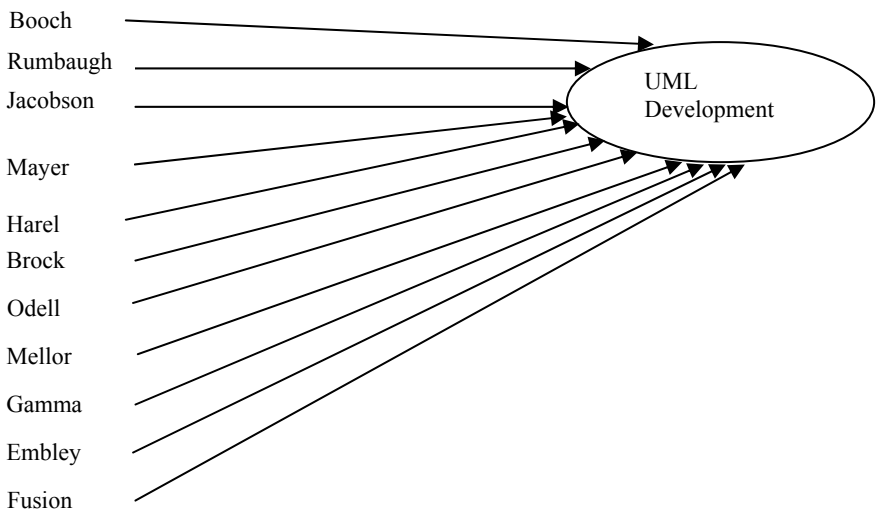

Figure 2: The Input for UML development

The major features of UML are:

- defined system structure for object modeling
- support for different model organization
- strong modeling for structure and behavior
- clear representation of concurrent activities
- support for object oriented patterns for design reuse.

The model for the object oriented development could be shown as in *Figure 3*. It could be classified as static/dynamic and logical/physical model.

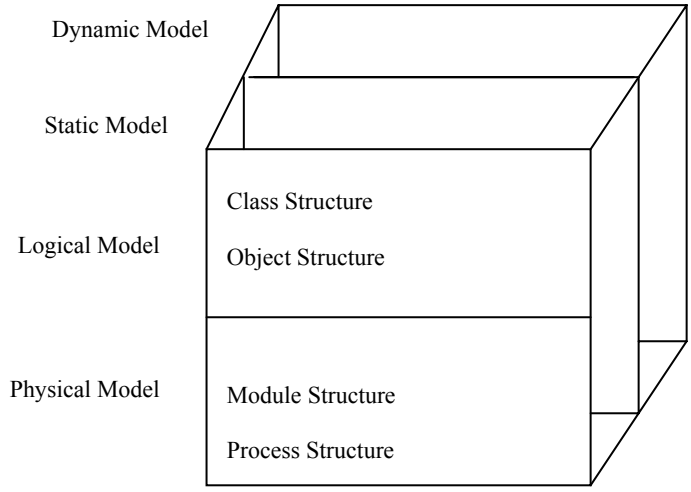

Figure 3: The Model for Object oriented development

The Logical view of a system serves to describe the existence and meaning of the key abstractions and the mechanism that form the problem space, or that define the system architecture.

The Physical model describes the concrete software and hardware components of the system's context or implementation.

UML could be used in visualizing, specifying, constructing and documenting object oriented systems. The major building blocks of UML are structural, behavioral, grouping, and annotational notations. Let us discuss these blocks, one by one.

a. **Structural Notations:** These notations include static elements of a model. They are considered as **nouns** of the UML model which could be conceptual or physical. Their elements comprises class, interface, collaboration, use case, active class, component, and node. It also includes actors, signals, utilities, processes, threads, applications, documents, files, library, pages, etc.

b. **Behavioral Notations:** These notations include dynamic elements of a model. Their elements comprises interaction, and state machine. It also includes classes, collaborations, and objects.

c. **Grouping Notations:** These notations are the boxes into which a model can be decomposed. Their elements comprises of packages, frameworks, and subsystems.

d. **Annotational Notations:** These notations may be applied to describe, illuminate, and remark about any element in the model. They are considered as explanatory of the UML. Their elements comprised of notes which could be used for constraints, comments and requirements.

UML is widely used as it is expressive enough, easy to use, unambiguous and is supported by suitable tools.

3.3 OBJECT MODELING NOTATIONS: BASIC CONCEPTS

A **system** is a collection of subsystems organised to accomplish a purpose and described by a set of models from different viewpoints.

A **model** is a semantically closed abstraction of a system which represents a complete and self-consistent simplification of reality, created in order to better understand the system.

A **view** is a projection into an organisation and structure of a system's model, focused on one aspect of that system.

A **diagram** is a graphical presentation of a set of elements.

A **classifier** is a mechanism that describes structural and behavioral features. In UML the important classifiers are class, interface, data type, signals, components, nodes, use case, and subsystems.

A **class** is a description of a set of objects that share the same attribute, operations, relationships, and semantics. In UML, it is shown by a rectangle.

An **attribute** is a named property of a class that describes a range of values that instances of the property may hold. In UML, they are listed in the compartment just below that class name.

An **operation** is the implementation of a service that can be requested from any object of the class to affect behavior. In UML, they are listed in the compartment just below that class attribute.

The notation for class, attribute, and operations is shown as:

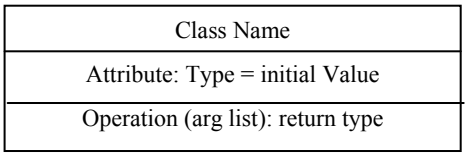

Figure 4: Class with attributes and operations

An **interface** is a collection of operations that are used to specify a service of a class or a component.

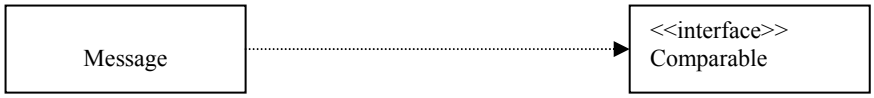

Figure 5: Realizing an interface

A **signal** is the specification of an asynchronous stimulus communicated between instances.

A **component** is a physical and replaceable part of the system that confirms to, and provides the realization of a set of interfaces. In UML, it is shown as a rectangle with tabs. The notation for component and interface is shown as:

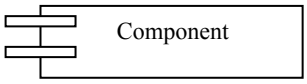

Figure 6: Component

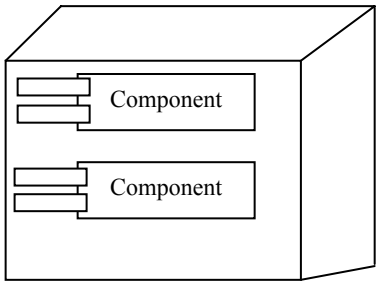

Figure 7: Components and Server

A **node** is a physical element that exists at runtime, and represents a computational resource generally having a large memory and often process capability. In UML, it is shown as a cube. The notation for node is shown as:

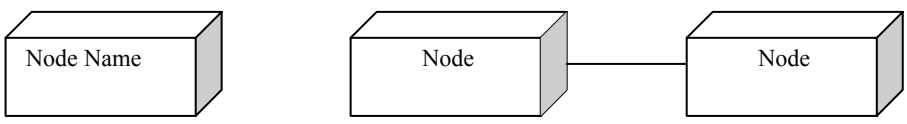

Figure 8: Node and relationship between nodes

A **use case** is a description of a set of sequence of actions that a system performs to yield an observable result that is of a value to an actor. The user is called actor and the process is depicted by use case. The notation for use case is shown as:

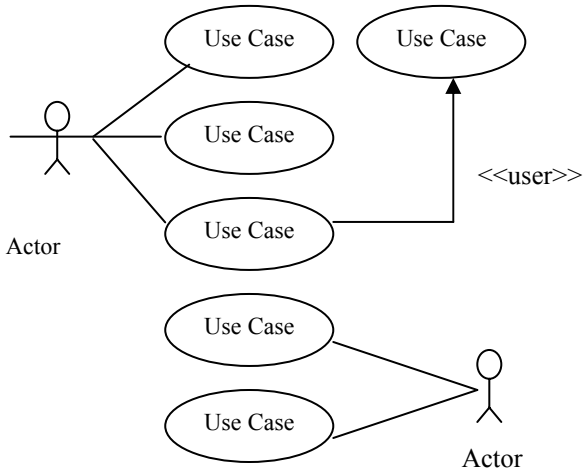

Figure 9: Relationship between actor and use case

A **subsystem** is a grouping of elements of which some constitute a specification of the behavior offered by other contained elements.

An **object** is an instance of a class. The object could be shown as the instance of a class, with the path name along with the attributes. Multiple objects could be connected with links. The notation for unnamed and named objects, object with path name, active objects, object with attributes, multiple objects, and self linked object is shown as:

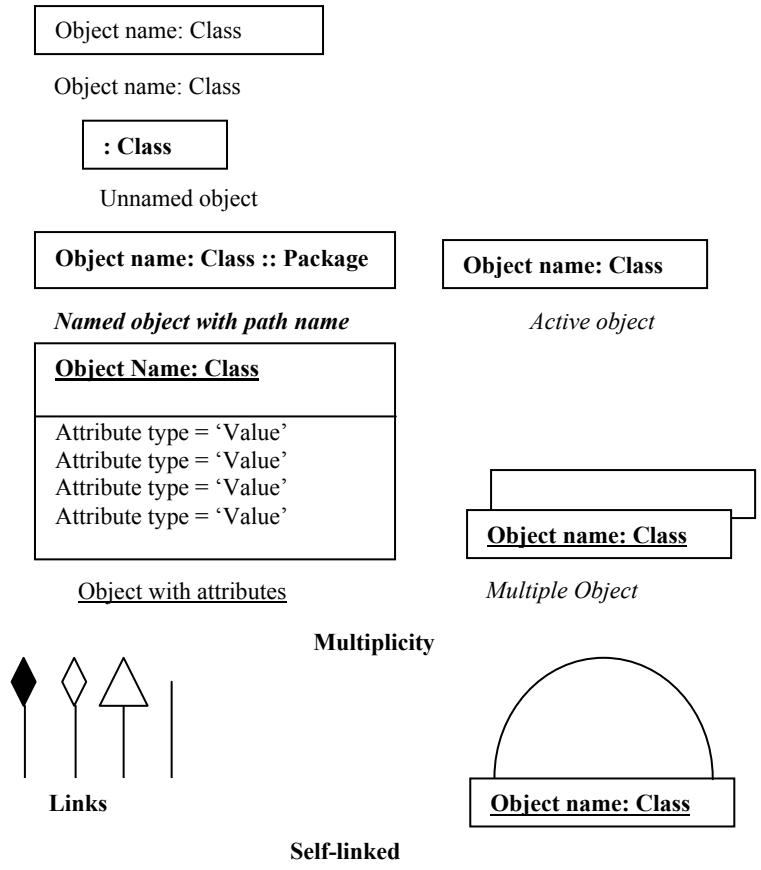

Figure 10: Different types of objects

A **package** is a general purpose mechanism for organising elements into groups. It can also contain other packages. The notation for the package shown contains name

and attributes as shown in *Figure11*. Packages are used widely in a Java based development environment. You may refer to the Unit 3 of Block 2 of the MCS-024 course for more details about packages.

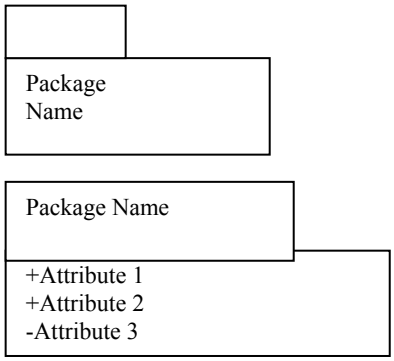

Figure 11: Package Diagram

A **collaboration** is a society of classes, interfaces and other elements that work together to provide some cooperative behavior that is bigger than the sum of its parts.

A **relationship** is a connection among things. In object models, the common types of relationships are inheritance, dependency, aggregation, containment, association, realisation, and generalisation. The notation for relationship between use cases is shown as:

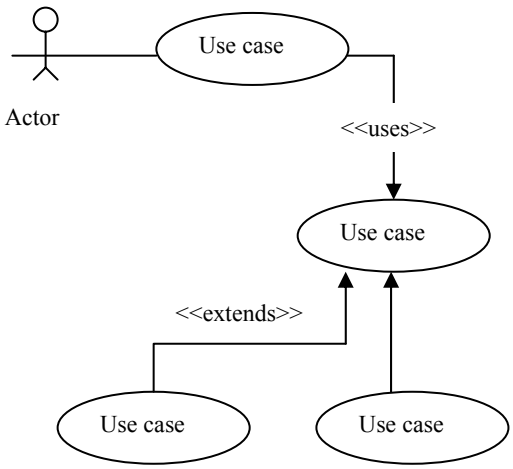

Figure 12: Relationship between different use case

Relationships exists in many forms. The notation for different form of relationship is shown as:

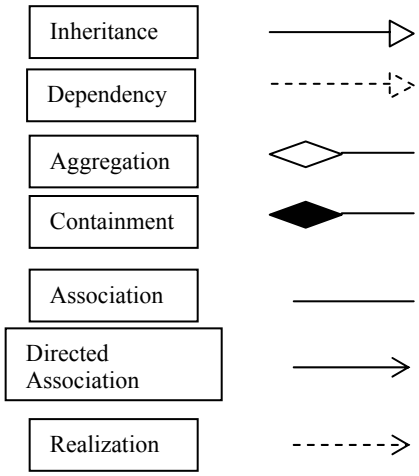

Figure 13: Common relationship types

A **dependency** is a relationship that states, that a change in specification of one thing may affect another thing, but not necessarily the reverse. In UML, it is shown as a dashed directed line. The notation for dependency between components and packages is shown as:

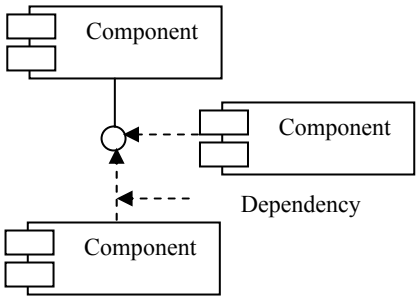

Figure 14: Dependency between components

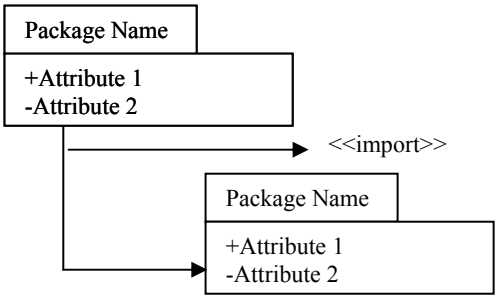

Figure 15: Dependency between packages

A **generalization** is a relationship between a general thing and a specific kind of thing. It is also called "is-a-kind-of" relationship. Inheritance may be modeled using generalization. In UML, it is shown as a solid directed line with a large open arrow pointing to the parents.

An **association** is a structural relationship that specifies that the objects of one thing are connected with the objects of another. In UML, it is shown as a solid line connecting same or different class. The notation for association between nodes is shown as:

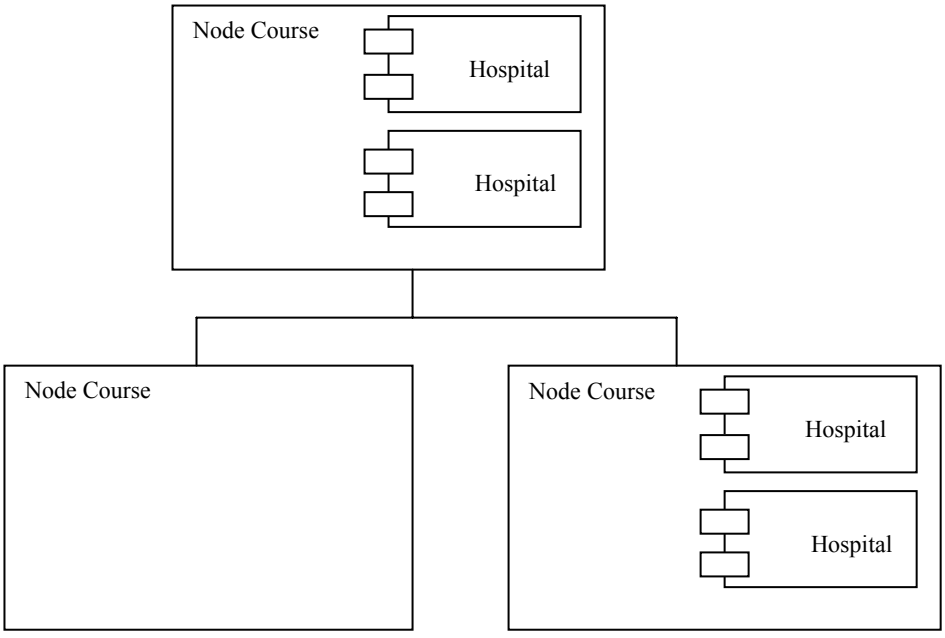

Figure 16: Association between nodes

The four enhancements that apply to association are name, role, multiplicity, and aggregation. Each class participating in an association has a specific role which is specified at the rear end of the association.

Multiplicity specifies how many objects may be connected across an instance of an association which is written as a range of values (like 1..*). The notation for roles and multiplicity between classes is shown as:

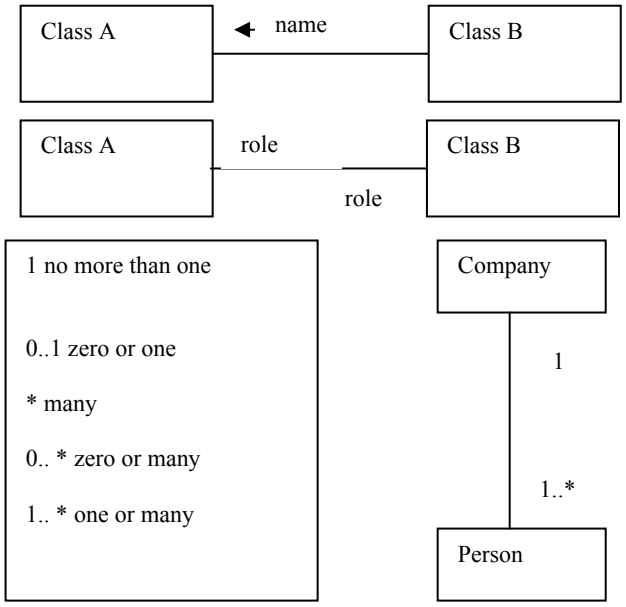

Figure 17: Various roles and multiplicity defined with association

An **aggregation** is a structural relationship that specifies that one class represents a large thing which constitute of smaller things. It represents "has-a" relationship. In UML, it is shown as association with an open diamond at the large end. The notation for aggregation is shown as:

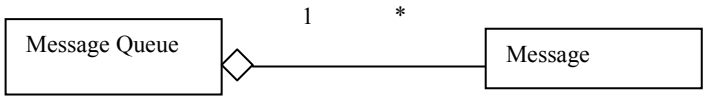

Figure 18: Aggregation

A **state** encompasses all the properties of the object along with the values of each of these properties.

An **instance** is a concrete manifestation of an abstraction to which a set of operations can be applied and which has a state that stores the effect of the operation.

A **transition** is a relationship between two states indicating that an object in the first state will perform certain action and enter the second state when a specific event occurs and specific conditions are satisfied.

A **note** is a graphical symbol for rendering constraints or comments attached to an element or collection of elements.

☞ Check Your Progress 1

1) Which of the following is not a valid Architectural Definition language?

   a) Rapide b) ACME
   c) UML d) Pascal

   ……

   ……...

2) OMT is

a) Object Methodology Gateway
b) Objective Methodology Gateway
c) Object Management Gateway
d) Object Management Group

……

……..……..

3) Object Oriented Software Engineering is given by

a) Booch
b) Rumbaugh
c) Jacobson
d) None of these

……

…..……..

4) Booch was strong in

a) Analysis
b) Design
c) Implementation
d) Software engineering

……

5) Which of the following is not a valid UML notations?

a) Behavioral
b) Grouping
c) Transactional
d) Annotational

……

……

For object modeling some standard notations are used. Now let us discuss these basic notations. A well-defined logical notation is important in the software development process. It helps the software architect to clearly establish the software architecture and implement module integration. In order to define the commonly used diagrams in UML, it is essential to understand the basic concepts of the object modeling.

3.4 STRUCTURAL DIAGRAMS

The main purpose of structural diagram is to visualize, specify, construct and document the static aspects of a system. Their elements comprised of class, interface, active class, component, node, processes, threads, applications, documents, files, library, pages etc. The four main structural diagrams are class, object, component and deployment diagram.

3.4.1 Class Diagram

A class diagram is used to support functional requirement of system. In a static design view, the class diagram is used to model the vocabulary of the system, simple collaboration, and logical schema. It contains sets of classes, interfaces, collaborations, dependency, generalization and association relationship. The notation for classes and the relationship between classes is shown as:

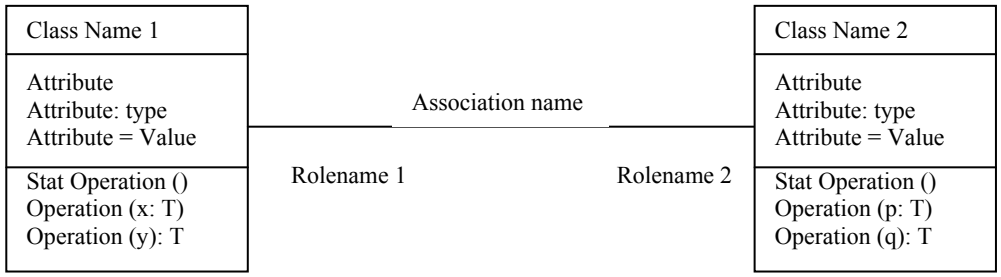

Figure 19: Relationship among classes

If in any college, there are limited classrooms that have to be allocated to different classes and instructors are fixed for all classes, then the class diagram for the allocation of classrooms and instructors is shown as:

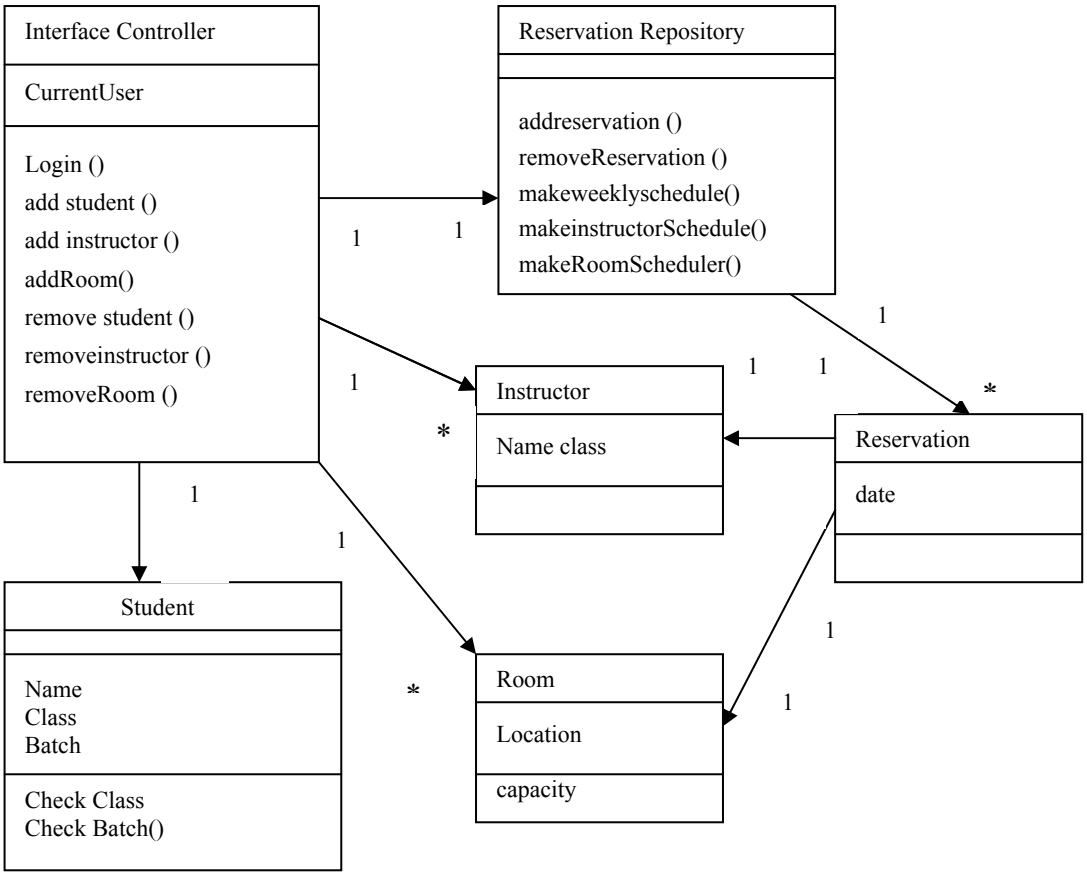

Figure 20: Class diagram for a class room scheduling system

3.4.2 Object Diagram

An object diagram shows a set of objects and their relationships at a point of time. In a static view, the object diagram is used to model interactions that consist of objects that collaborate the without any message passed among them. It contains name, graphical contents, notes, constraints, packages and subsystems. The notation for objects and the relationship between objects is shown as:

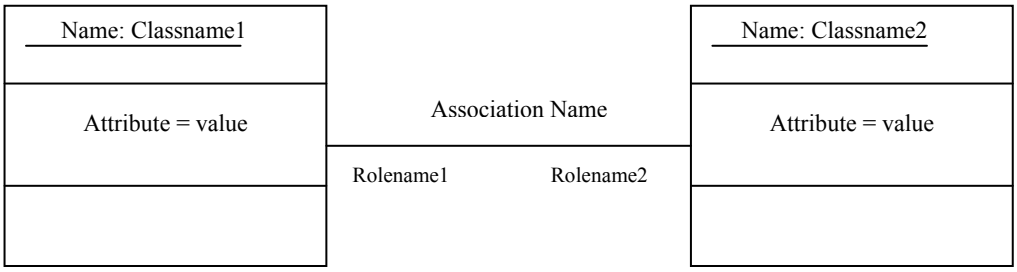

Figure 21: Relationship among objects

3.4.3 Component Diagram

A component diagram shows a set of component and their relationships. In a dynamic model, the component diagram is used to model physical components such as executable releases, libraries, databases, files or adaptable systems. It contains components, interfaces, packages, subsystems, dependency, generalization, association, and relationship. The notation for components and relationship between components is shown as:

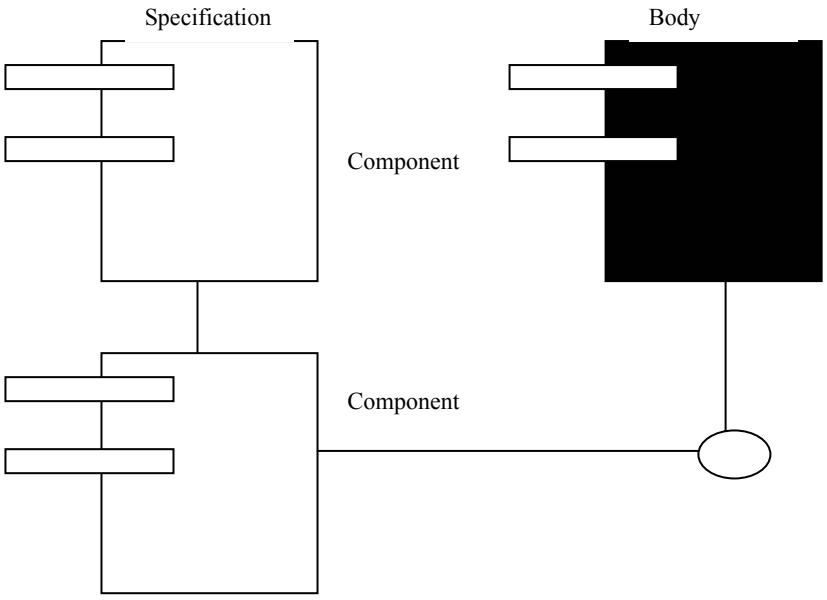

Figure 22: Relationship among components

The component diagram for ATM is shown as:

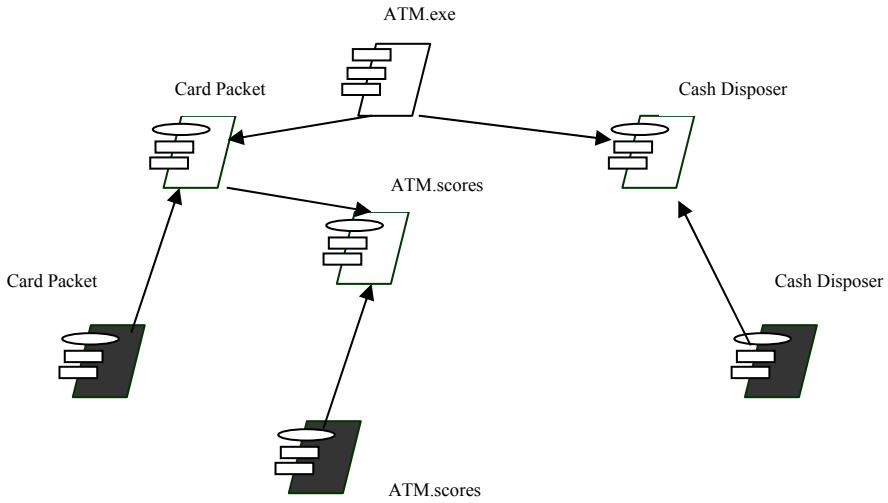

Figure 23: Component diagram for ATM

3.4.4 Deployment Diagram

A deployment diagram shows all the nodes on the network, their interconnections, and processor execution. In a dynamic model, a deployment diagram is used to represent computational resources. The notation for nodes and relationship between processors and devices is shown as:

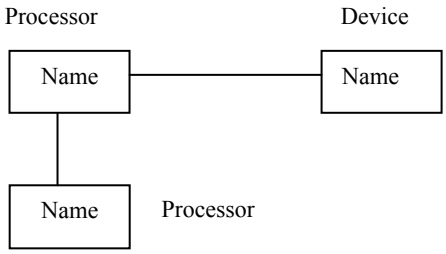

Figure 24: Relationship among nodes

The deployment diagram for student administration is shown as:

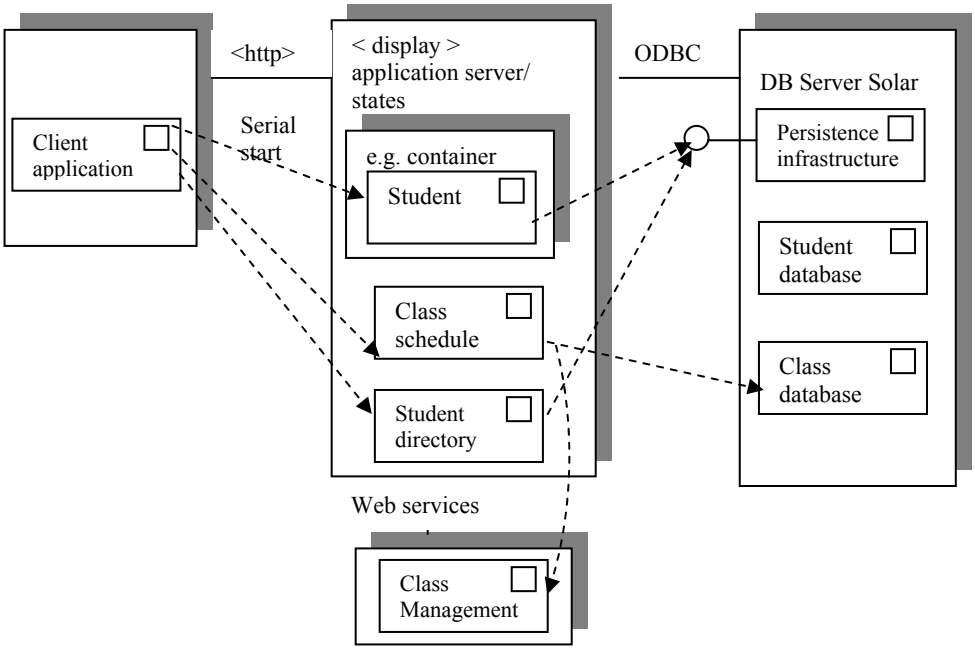

Figure 25: Deployment diagram for Student Administration

Now you are familiar with structured diagram. To represent dynamic aspect of the structured system, behavioral diagrams are used. In the next section, we will study various behavioral diagrams.

3.5 BEHAVIORAL DIAGRAMS

The main purpose of behavioral diagrams is to visualize, specify, construct, and document the dynamic aspects of a system. The interaction between objects indicating the flow of control among them is shown using these diagrams. The flow of control may encompass a simple, sequential thread through a system, as well as complex flows that involves branching, looping, recursion and concurrency. They could model time ordering (sequence diagram) or sequence of messages (collaboration diagram). The four main behavioral diagrams are: use case, interaction, activity and statechart diagram.

3.5.1 Use Case Diagram

A use case diagram shows a set of use cases, actors, and their relationships. These diagrams should be used to model the context or the requirement of a system. It contains use cases, actors, dependency, generalization, association, relationship, roles, constraints, packages, and instances. The use case diagram makes systems, subsystems, and classes approachable by presenting an outside view of how the elements may be used in context. The notation for use cases and relationship among use cases, base cases and extend cases is shown as:

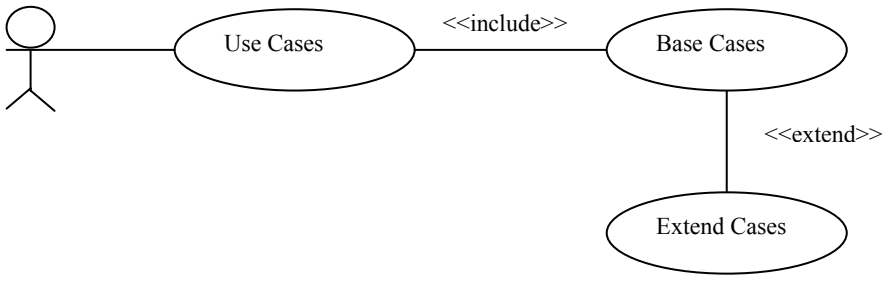

Figure 26: Relationship among use cases

3.5.2 Interaction Diagram

An interaction diagram shows an interaction, consisting of a set of objects and their relationships, including the messages that may be dispatched among them. These diagrams should be used to model the dynamic aspect of the system. It includes sequence diagrams and collaboration diagrams. Here we will discuss two interaction diagrams, sequence diagrams and collaboration diagrams.

3.5.2.1 Sequence Diagrams

A sequence diagrams are interaction diagrams that emphasize the time ordering of messages. In UML it is shown as a table that shows objects arranged along the X axis and messages, ordered in increasing time, along the Y axis. It has a global life line and the focus of control. An object life line is the vertical dashed line that represents existence of an object over a period of time. The focus of control is tall and thin rectangle that shows the period during which an object is performing an action. The notation for depiction of sequence among objects with certain conditions is shown as:

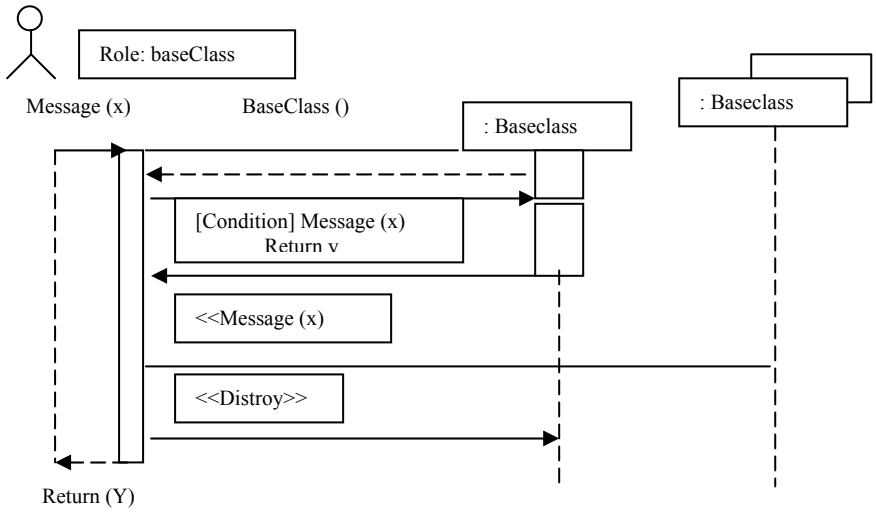

Figure 27: Sequence diagram

The sequence diagram for sending the document along the network is shown as:

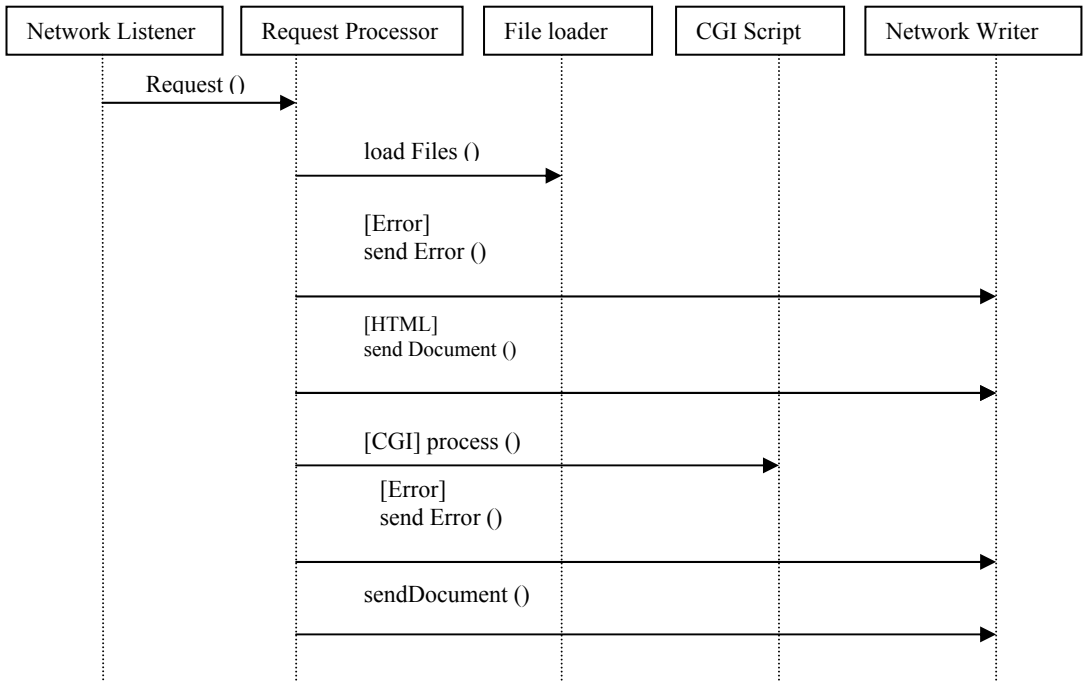

Figure 28: Sequence diagram for sending document

3.5.2.2 Collaboration Diagrams

Collaboration diagrams are interaction diagrams that emphasize the structural organisation of an object that send and receive messages. There is always a path in collaboration diagrams to indicate how one object is linked to another, and sequence numbers to indicate the time ordering of a message. The notation for depiction of a collaboration diagram is shown as:

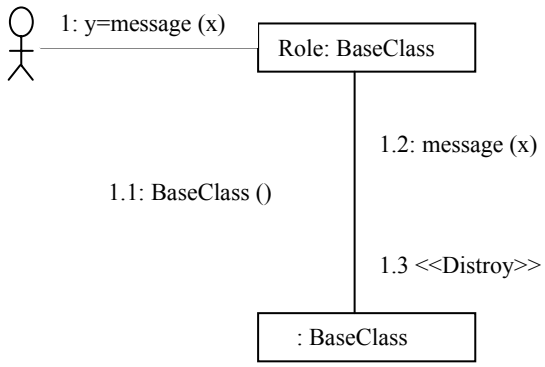

Figure 29: Collaboration diagram

The collaboration diagram for the execution using J2ME and EJB from the remote database is shown as:

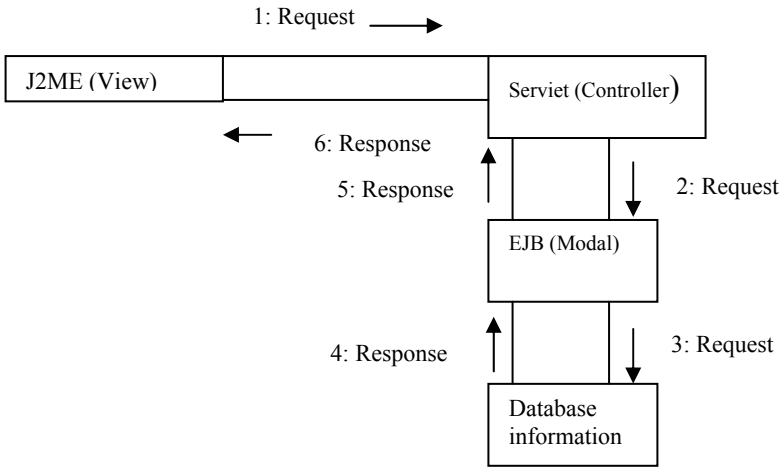

Figure 30: Collaboration diagram for execution using J2ME, Servlet and EJB

3.5.3 Activity Diagram

Activity diagrams show the flow from one activity to another. An activity is an ongoing non atomic execution within a state machine. Activity ultimately results in some action, which is made up of executable atomic computations that result in a change in state of the system, or the return of a value. It contains activity states, action states, transition states, and objects. The activity diagram for encryption of a message send through e-mail is shown as:

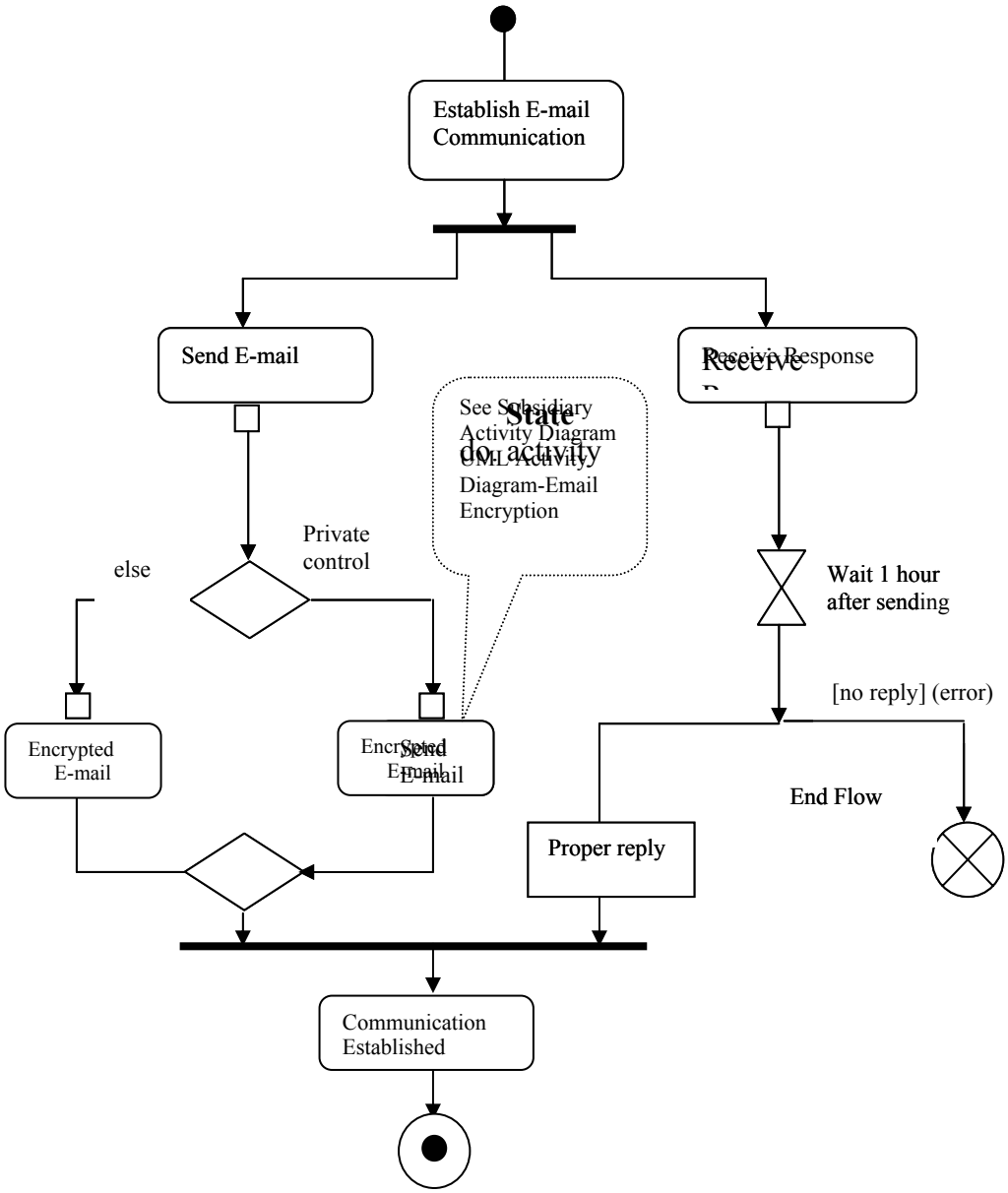

Figure 31: Activity diagram for E-mail encryption

3.5.4 Statechart Diagram

A state chart shows a state machine, emphasizing the flow of control from one state to another. A state machine is a behaviour that specifies the sequence of states that an object goes through during the life time in response to events together with its response to those events. A state is a condition/situation during the object's life which performs some activity, or waits for some event. It contains simple states, composite states, transitions, events, and actions. The notation for multiple states depending on events/action based on set of activities is shown as:

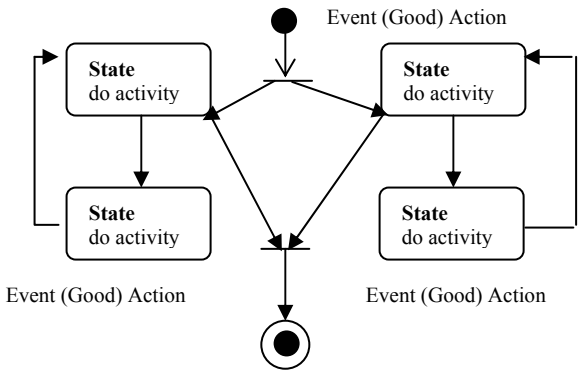

Figure 32: State diagram for multiple activities

☞ Check Your Progress 2

1) In a class diagram, a class is denoted by

a) rectangle b) circle
c) ellipse d) oval

……

……

2) An actor in use case diagrams is a

a) process b) subprogram
c) users d) comments

……

……

3) Which of the following diagram have to be numbered to understand their order?

a) Object b) Collaboration
c) Component d) Deployment

……

……

4) Which of the following diagram is used for physical layer?

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

……

……

5) A particular state in a statechart diagram is denoted by

a) rectangle b) circle
c) ellipse d) oval

……

……

6) A bull's eye icon is used to represent _____ in a statechart diagram

a) initial state b) action
c) final state d) event

……

……

7) Which of the following diagram's is used for dynamic modeling?

a) class b) object
c) use case d) interaction

……

……

8) Which view plays a special role integrating the contents of other views?

a) Use case view b) Process view
c) Design view d) Implementation view

……

……

……

3.6 MODELING WITH OBJECTS

A model is an abstract representation of a specification, design or system from a particular view. A modeling language is a way of expressing the various models produces during the development process. It is a collection of model elements. It is normally diagrammatic. It has

syntax – the rules that determine which diagrams are legal
semantics – the rules that determine what a legal diagram means.

A model is a semantically closed abstraction of a system compose of elements. It could be visualized using any of the following five views:

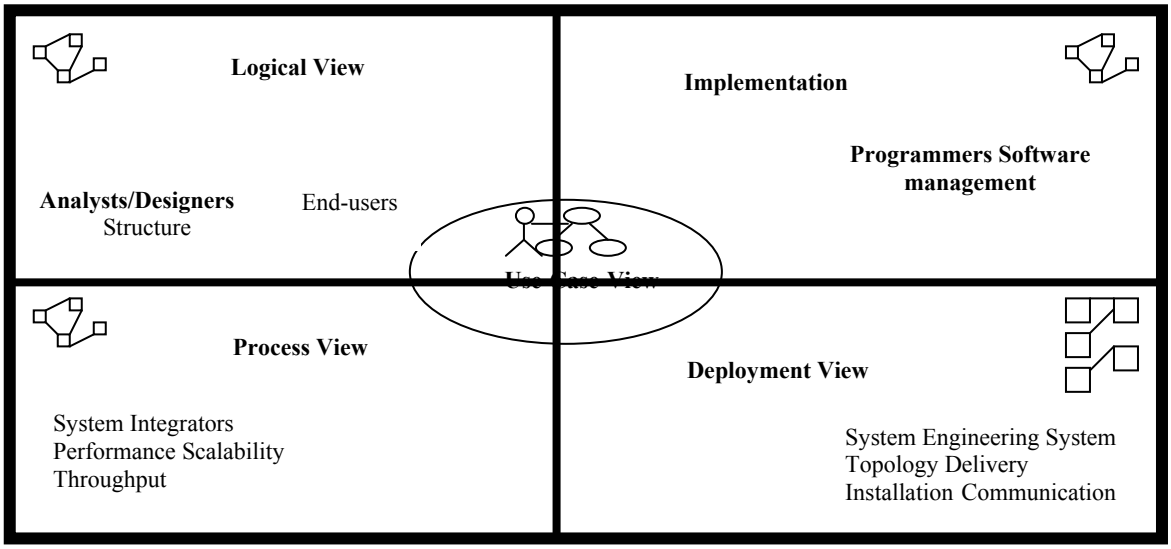

Figure 33: 4+1 View of Software Architecture

a. **Logical view** — This view is concerned with the functional requirements of the system. It is used early in the elaboration phase with the creation of class and packages, using a class diagram which may reflect the strategic dimension of the system.

b. **Implementation view** — This view focuses on the actual software module organisation within the developmental environment. It includes taking the derived requirement, software management, reuse, and constraints imposed by the program tools. The physical partitioning is done in this phase.

c. **Process view** — This involves the runtime implementation structure of the system. It contains requirements such as performance, reliability, scalability, integrity, synchronization, etc. Executable components are used here to show runtime components to map classes such as java applet, activeX component or DLL.

d. **Deployment view** — This view demonstrates mapping software to process nodes showing the configuration of runtime processing elements. It takes into account, requirements such as availability, reliability, performance and scalability. Major issues here are processor, architecture, speed, along with inter process communication, bandwidth and distributed facilities.

e. **Use case view** This view addresses and validates the logical, process, component, and deployment view.

For an iterative and incremental life cycle, the two criteria are time and process. The major components of showing a project development along with time scale, inception, elaboration, construction and transition. When the project is structured along with the process scale, then the major steps are business modeling, requirements, analysis and design, implementation, testing and deployment. The mapping between time and process scale can be shown diagrammatically for more clarity.

Using UML, it is possible to generate code in any programming language from UML model (called forward engineering) and reconstruct a model from an implementation into UML (called reverse engineering) as show in the *Figure 34* given below.

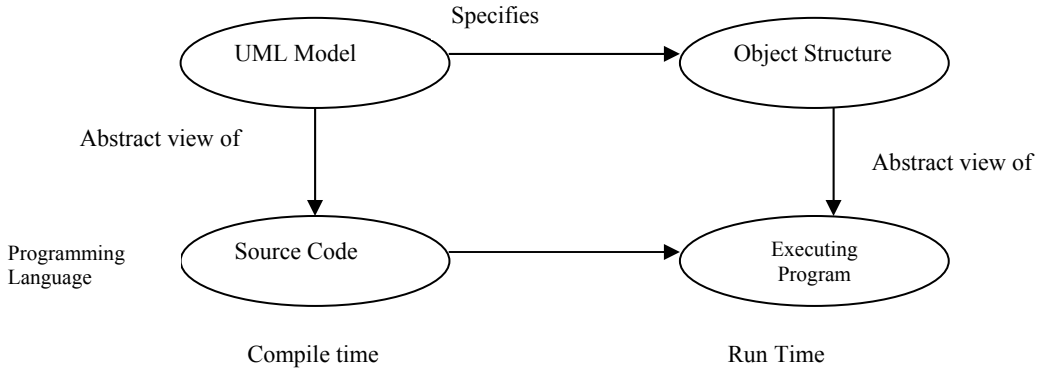

Figure 34: The relationship between models and code

☞ Check Your Progress 3

1) Create a use case diagram for a cell phonebook.

……

…..…….

2) Create a sequence diagram for a logon scenario.

……

…..…….

3.7 SUMMARY

This Unit provides an introduction to the basic concept of object modeling notations. The major diagrams used with UML have been discussed. An idea has been provided about different views and the corresponding diagrams. This Unit only contains the introduction of various diagrams, and the student is not expected to be an expert in designing every diagram.

3.8 SOLUTIONS /ANSWERS

Check Your Progress 1

1. d) 2. d) 3. c) 4. b)

Check Your Progress 2

1. a) 2. c) 3. b) 4. d)

5. d) 6 .c) 7.d) 8. a)

Check Your Progress 3

1)

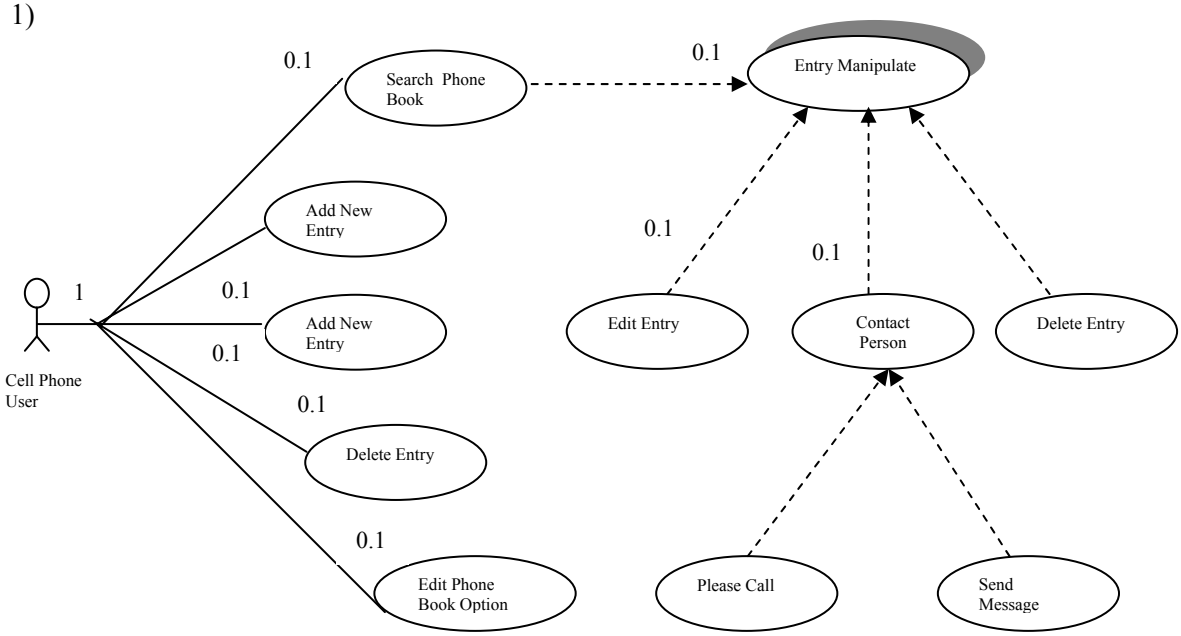

Figure 35: Use case diagram for a cell phonebook

2)

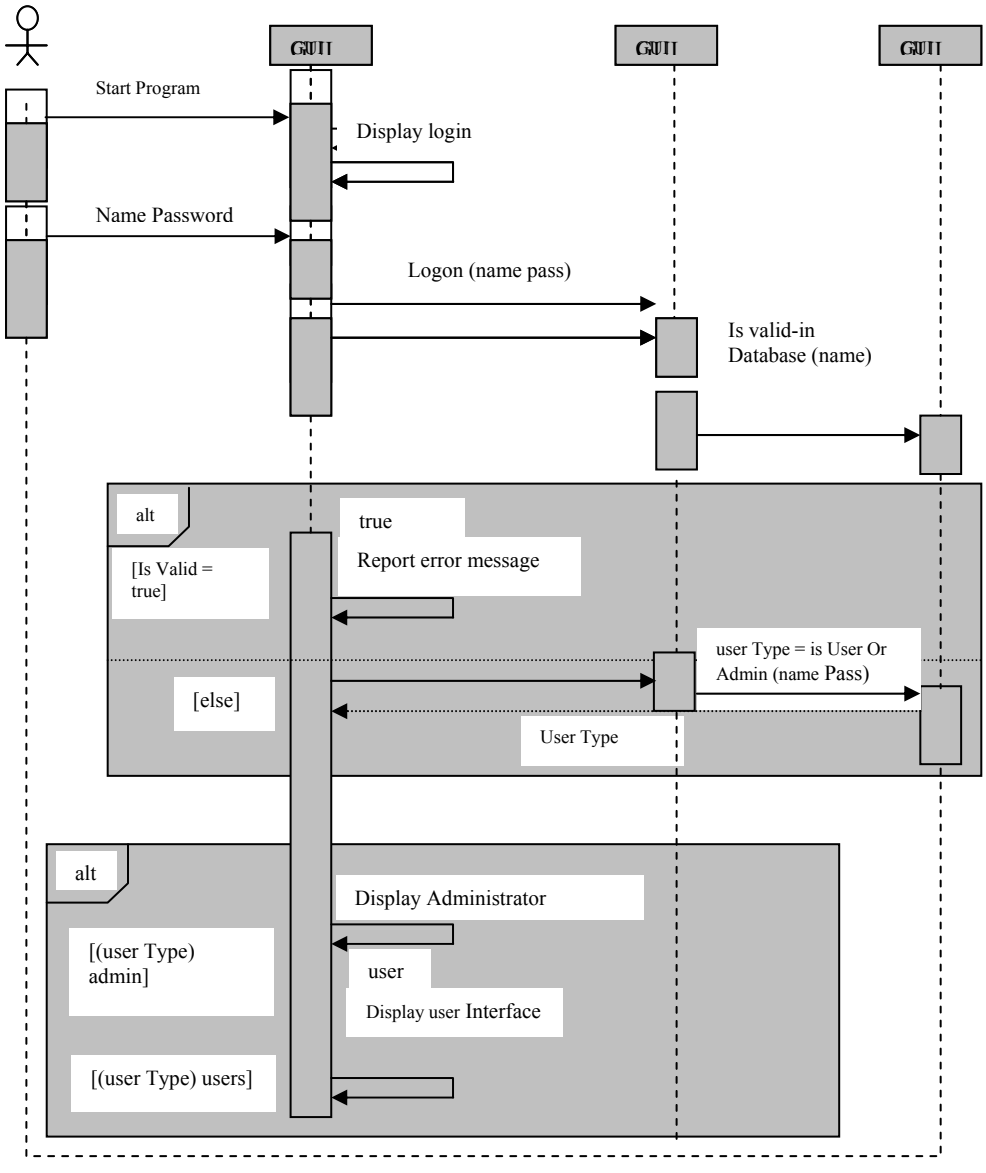

Figure 36: Sequence Diagram for logon scenario