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

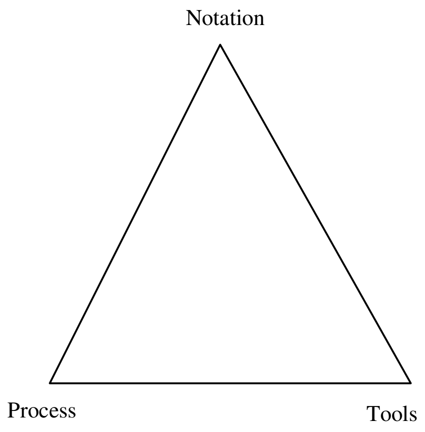

Figure 1: Components of project implementation

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.

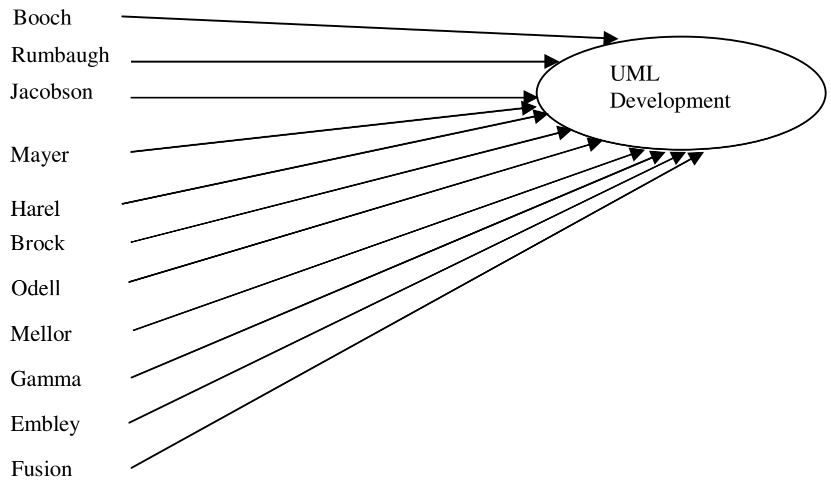

Figure 2: The Input for UML development

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.

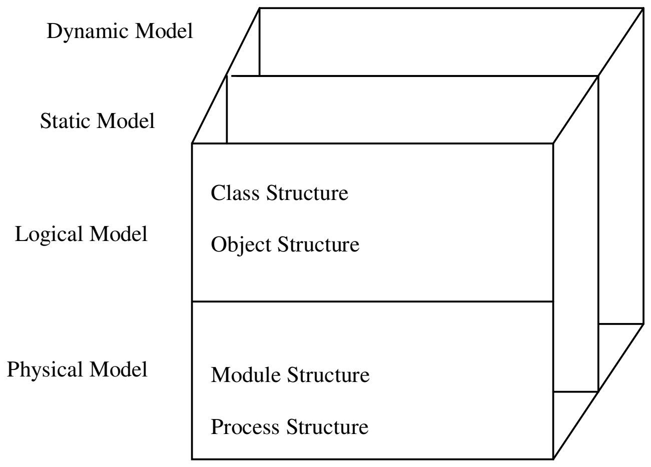

Figure 3: The Model for Object oriented development

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:

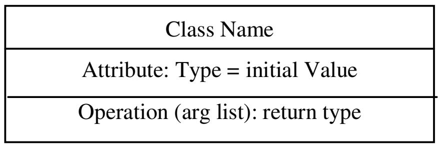
Class Name
Attribute: Type = initial Value
Operation (arg list): return type

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.

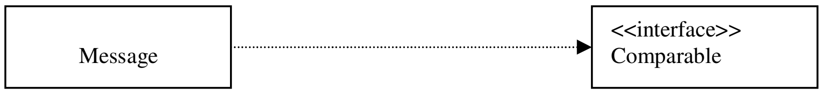

Figure 5: Realizing an interface

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:

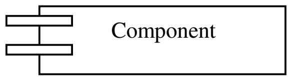

Figure 6: Component

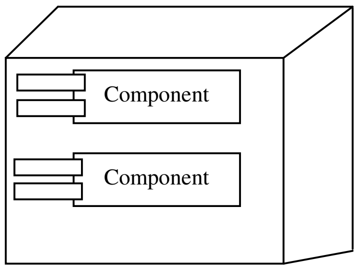

Figure 7: Components and Server

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:

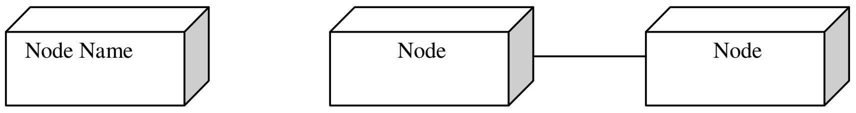

Figure 8: Node and relationship between nodes

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:

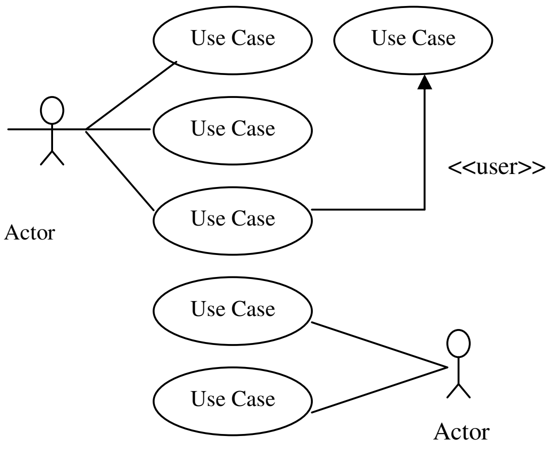

Figure 9: Relationship between actor and use case

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:

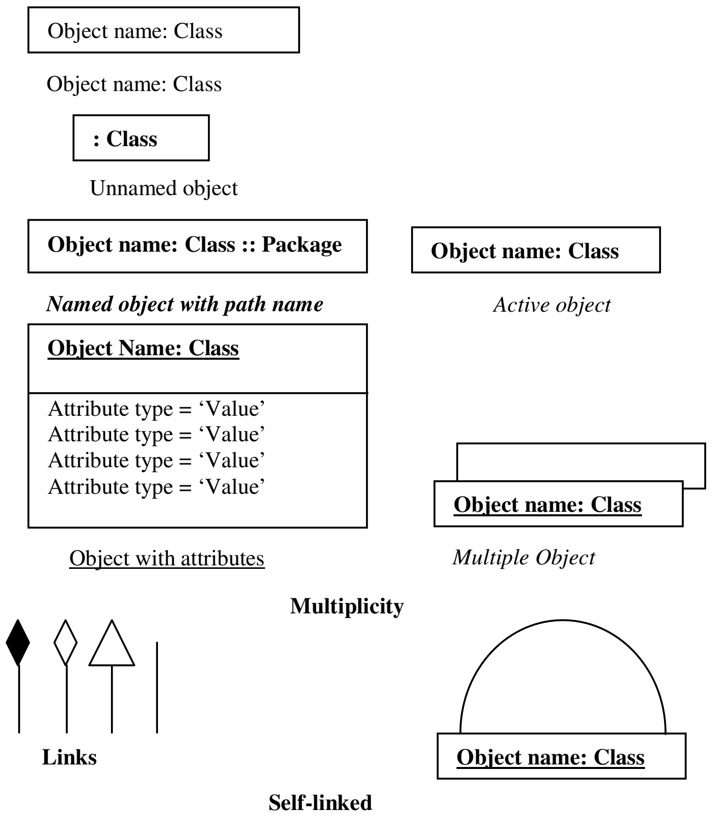

Figure 10: Different types of objects

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.

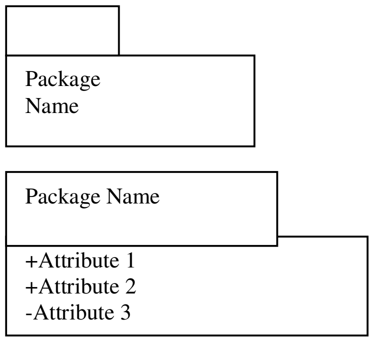

Figure 11: Package Diagram

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:

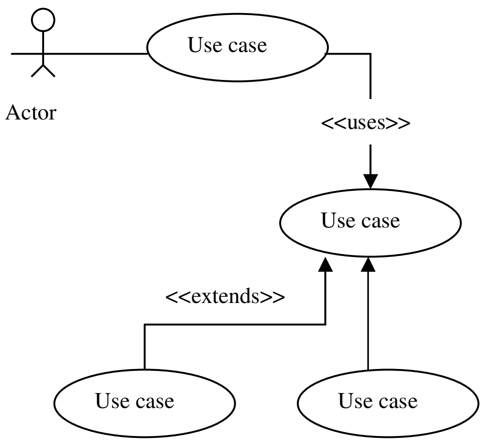

Figure 12: Relationship between different use case

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

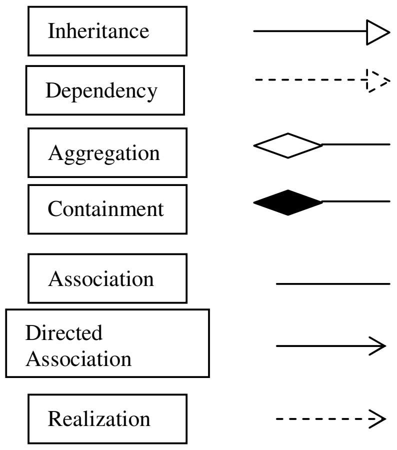

Figure 13: Common relationship types

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:

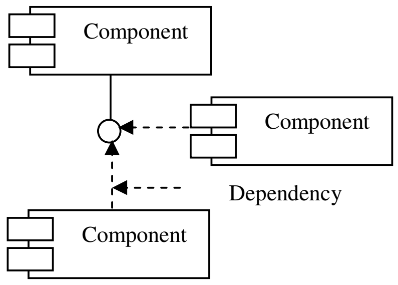

Figure 14: Dependency between components

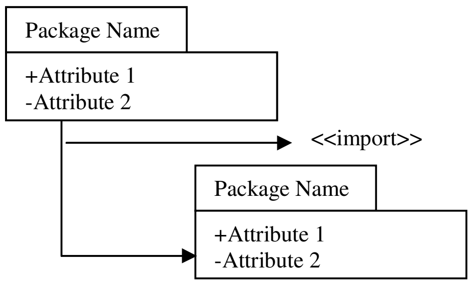

Figure 15: Dependency between packages

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:

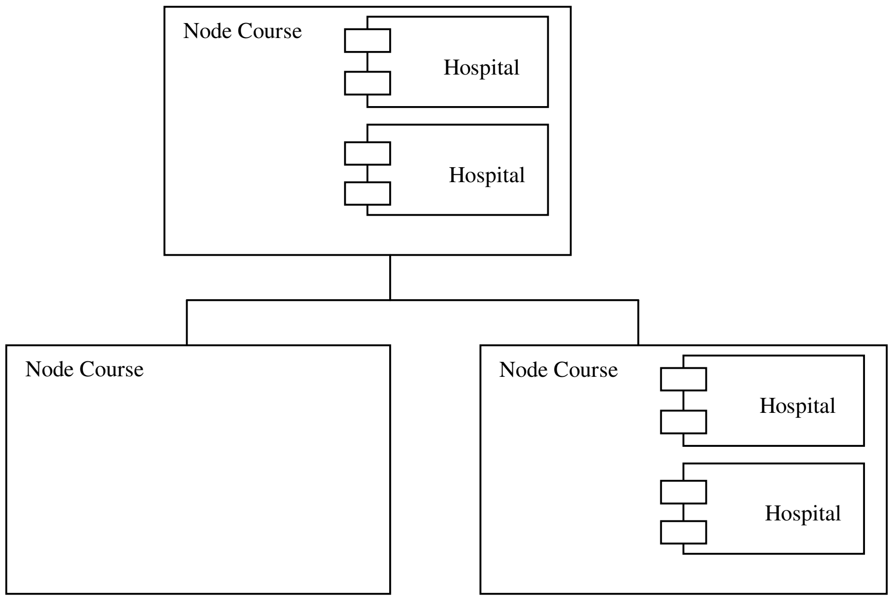

Figure 16: Association between nodes

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:

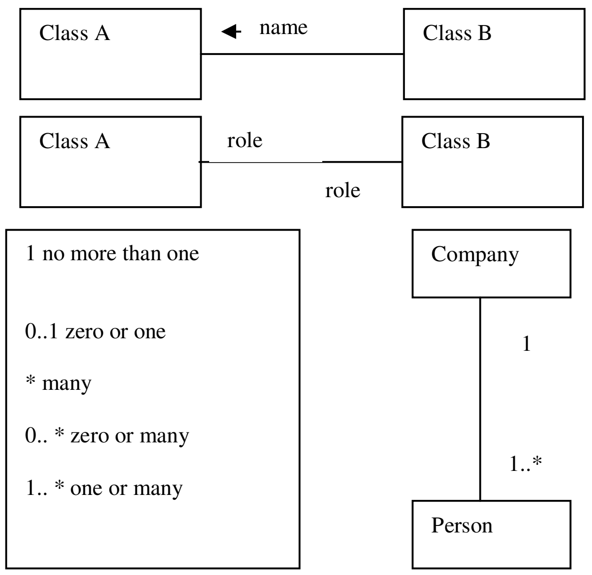

Figure 17: Various roles and multiplicity defined with association

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:

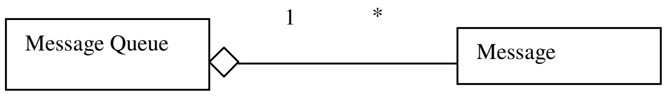

Figure 18: Aggregation

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:

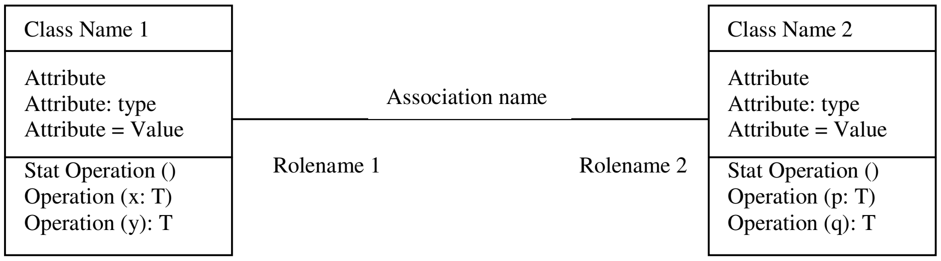

Figure 19: Relationship among classes

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:

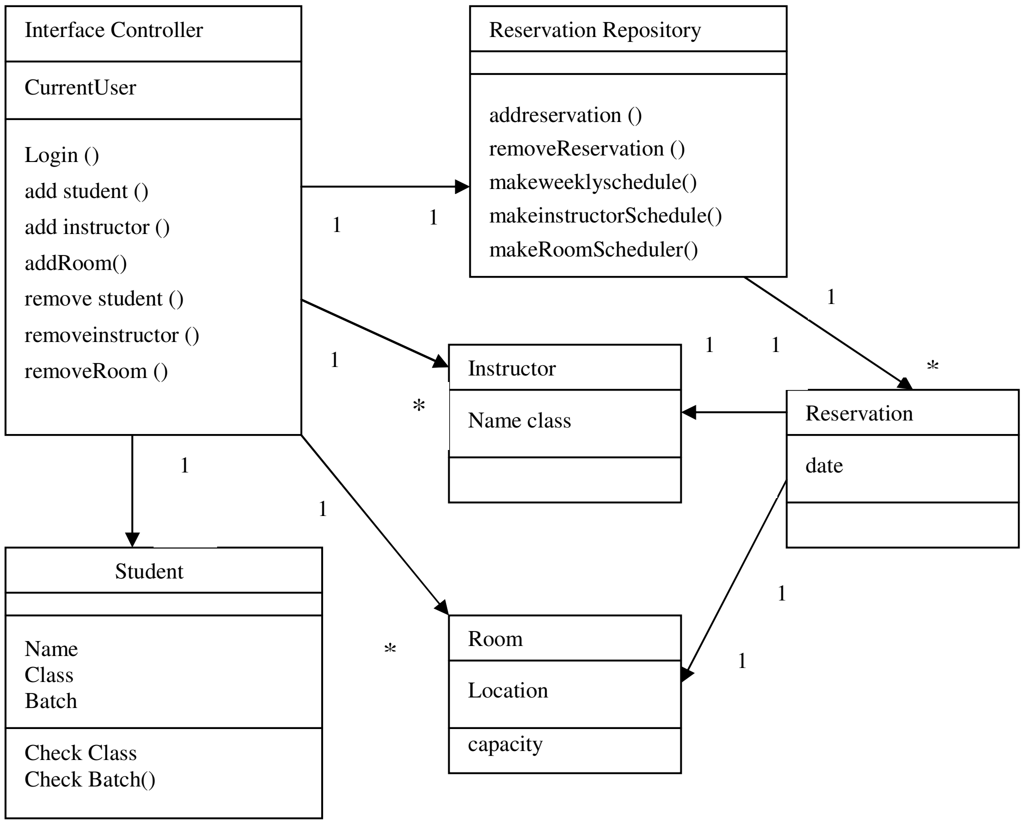

Figure 20: Class diagram for a class room scheduling system

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:

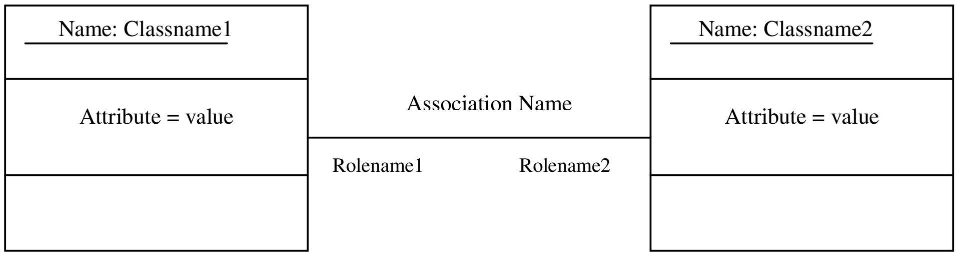

Figure 21: Relationship among objects

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:

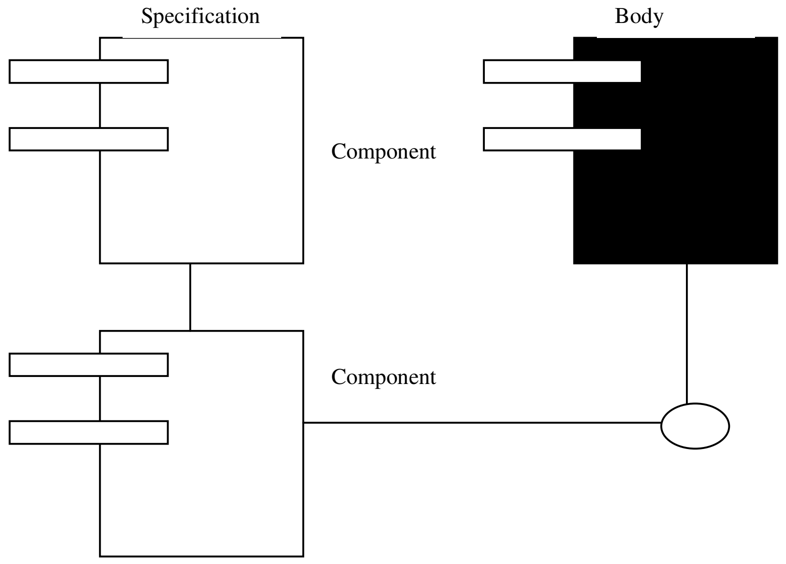

Figure 22: Relationship among components

The component diagram for ATM is shown as:

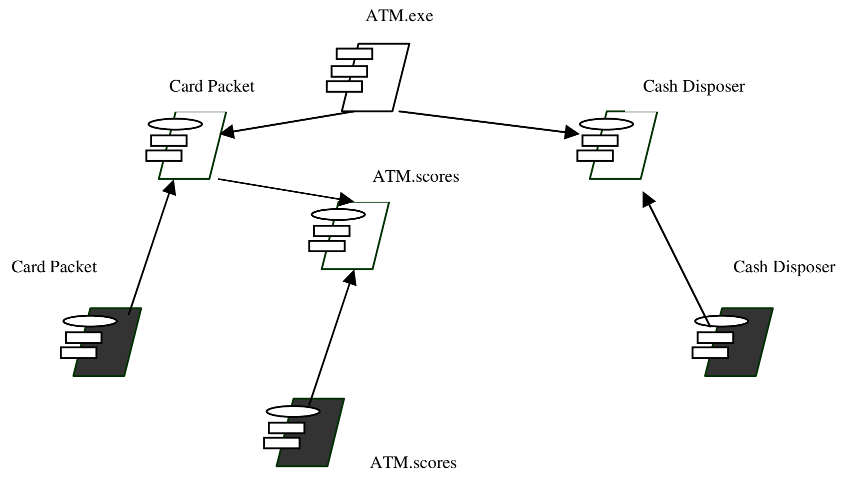

Figure 23: Component diagram for ATM

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:

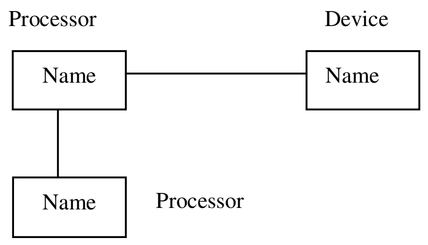

Figure 24: Relationship among nodes

The deployment diagram for student administration is shown as:

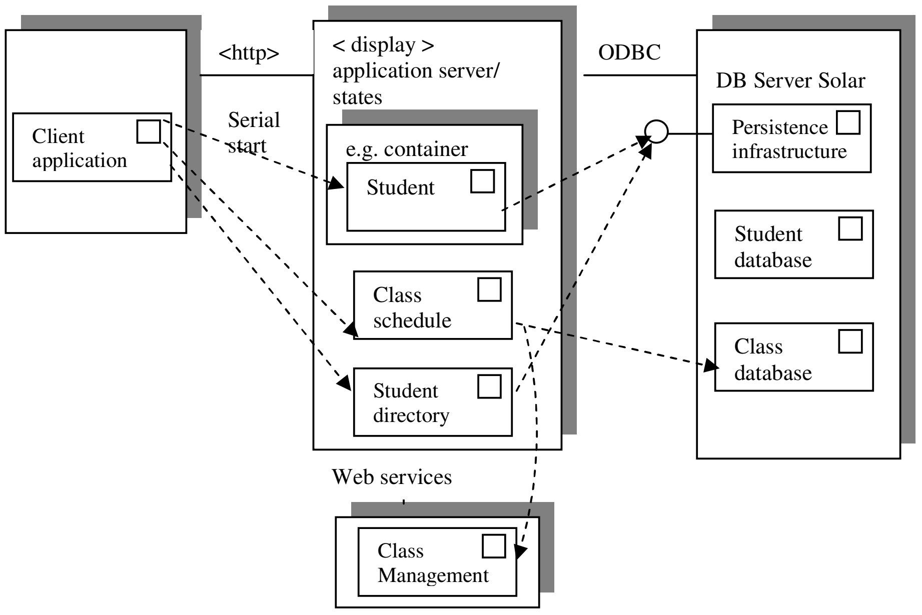

Figure 25: Deployment diagram for Student Administration

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:

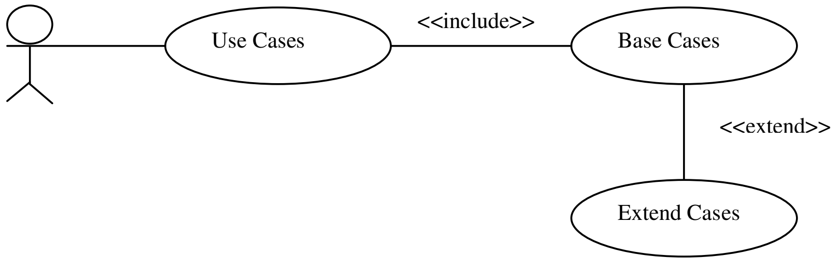

Figure 26: Relationship among use cases

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:

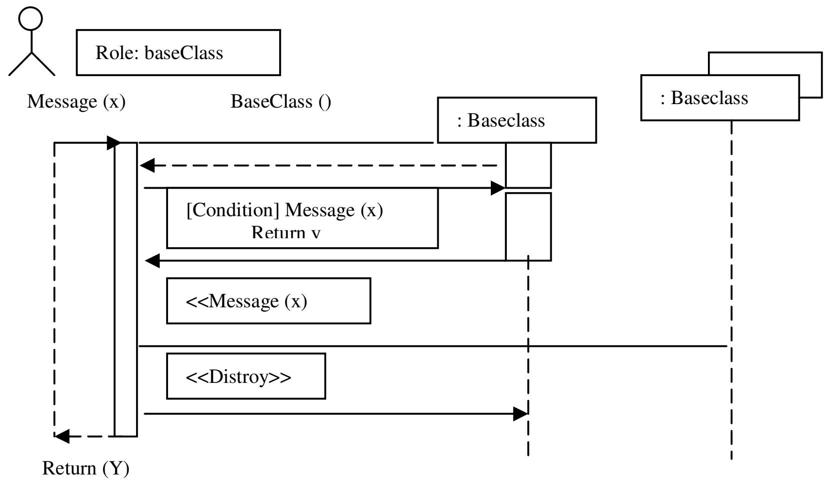

Figure 27: Sequence diagram

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

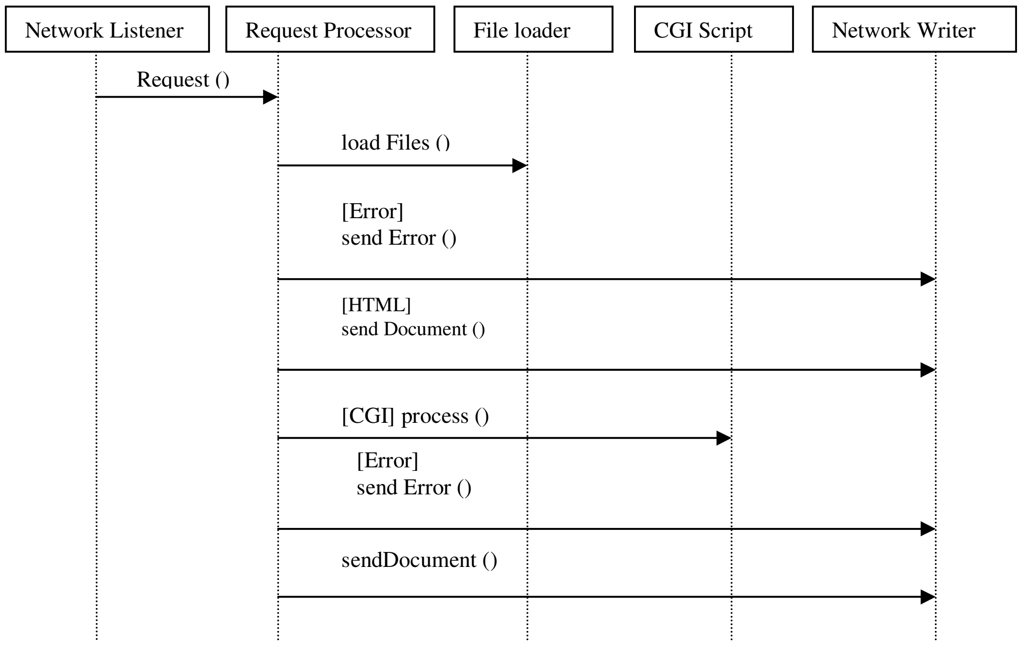

Figure 28: Sequence diagram for sending document

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:

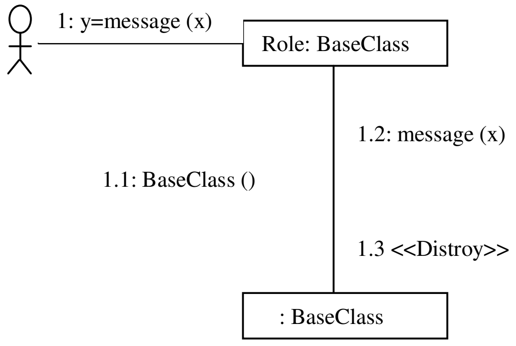

Figure 29: Collaboration diagram

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

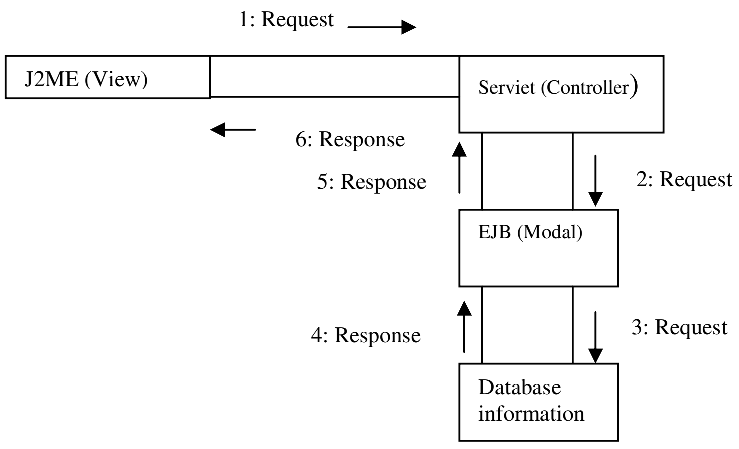

Figure 30: Collaboration diagram for execution using J2ME, Servlet and EJB

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:

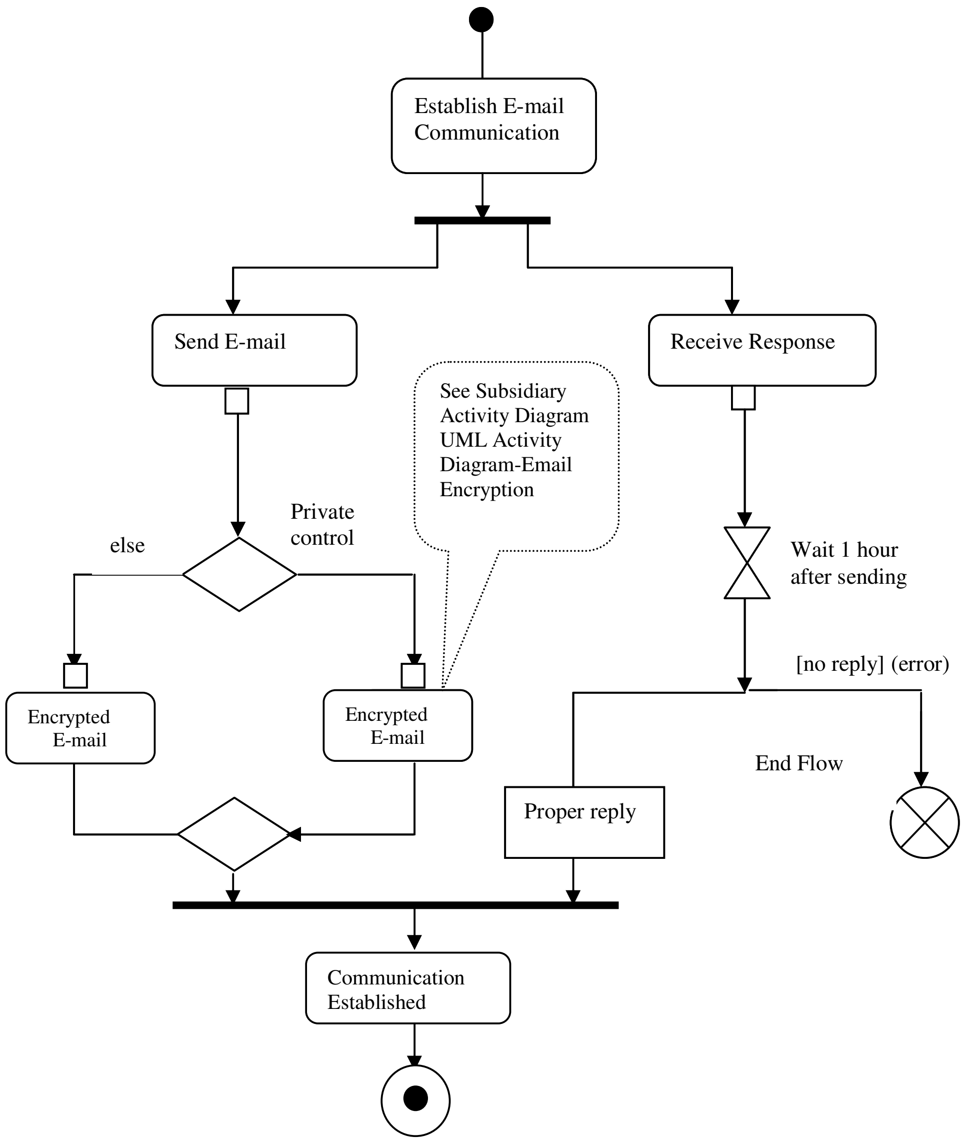
Establish E-mail Communication
Send E-mail
Receive Response
See Subsidiary Activity Diagram UML Activity Diagram-Email Encryption
Private control
else
Wait 1 hour after sending
[no reply] (error)
Encrypted E-mail
Encrypted E-mail
End Flow
Proper reply
Communication Established

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:

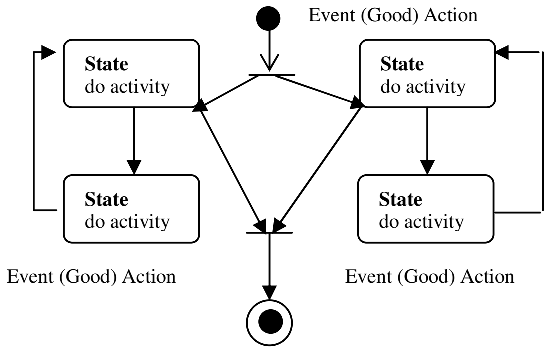

Figure 32: State diagram for multiple activities

☞ 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:

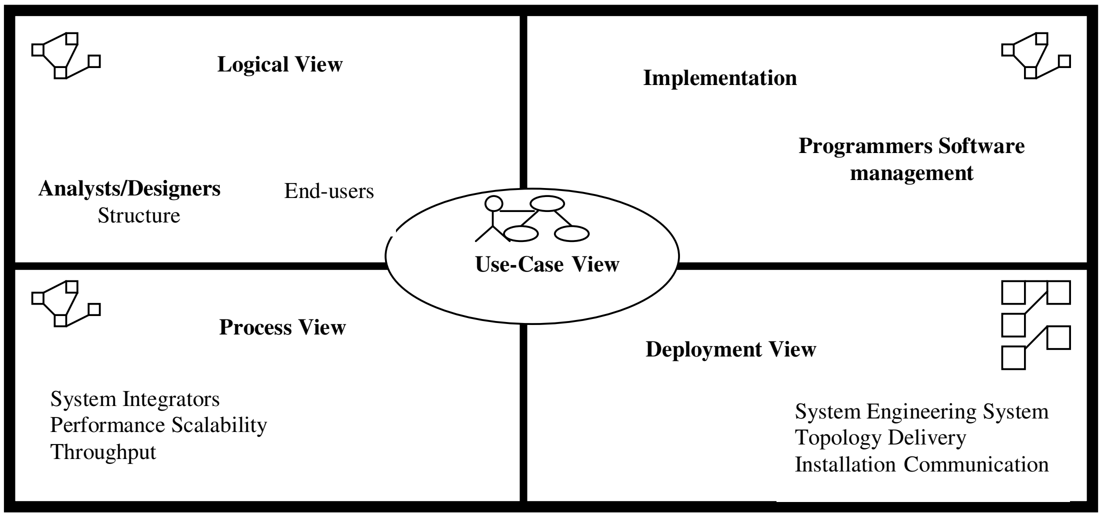

Figure 33: 4+1 View of Software Architecture

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.

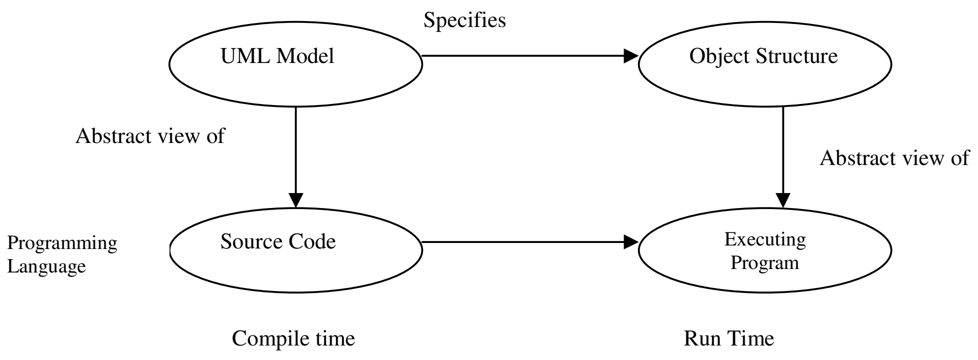

Figure 34: The relationship between models and code

☞ **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)

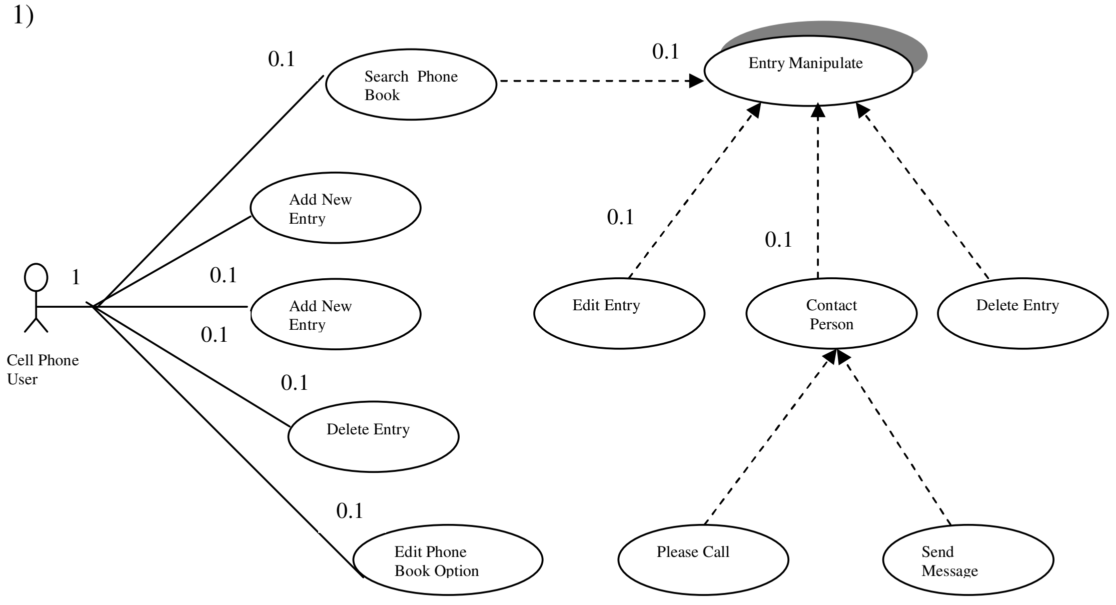

Figure 35: Use case diagram for a cell phonebook

2)

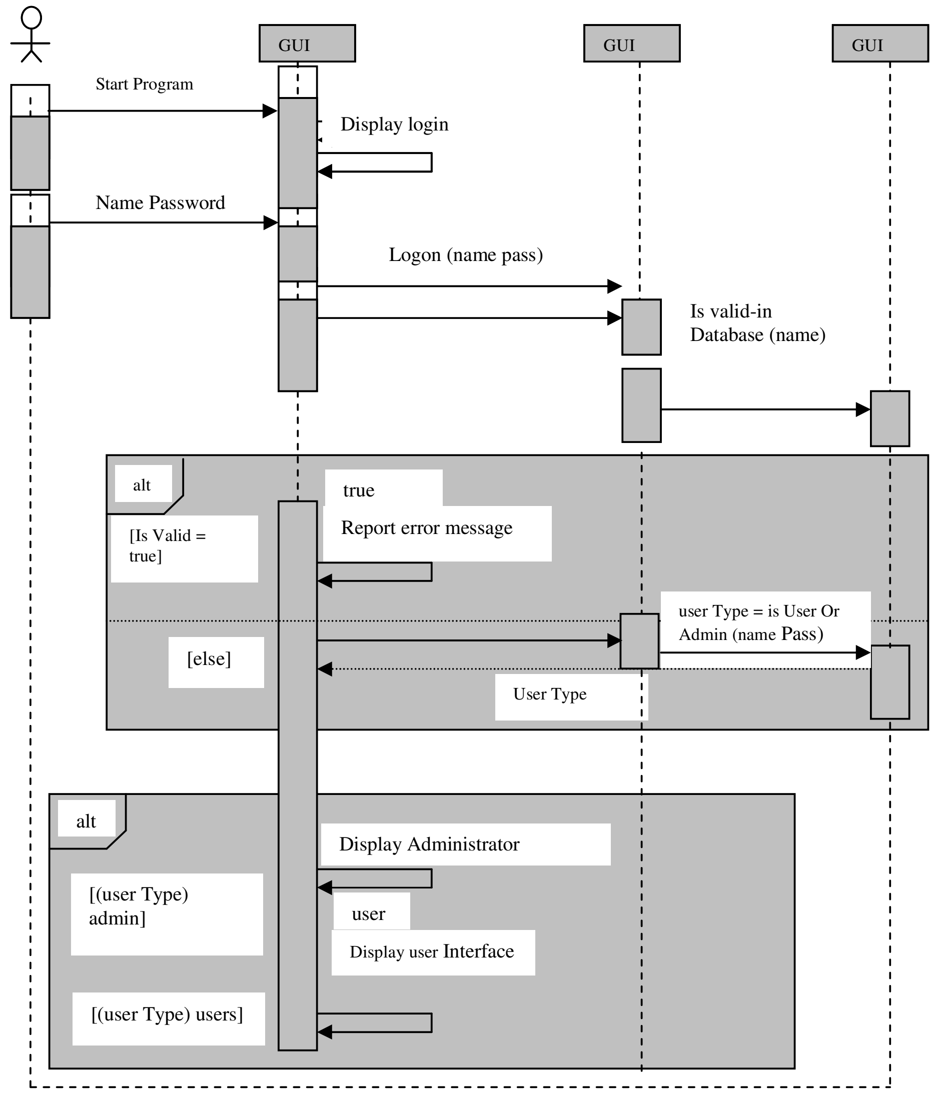

Figure 36: Sequence Diagram for logon scenario