ALAGAPPA UNIVERSITY

(Accredited with 'A+' Grade by NAAC (CGPA:3.64) in the Third Cycle
And Graded as Category-I University by MHRD-UGC)
(A State University Established by the Government of Tamilnadu)

KARAIKUDI – 630003

DIRECTORATE OF DISTANCE EDUCATION

MASTER OF COMPUTER APPLICATIONS

31533

OBJECT ORIENTED ANALYSIS AND DESIGN

Author:
Dr. N. Nagarajan
Assistant Professor
Department of Computer Applications
Alagappa University
Karaikudi-630003

Reviewer:
Dr. P. Prabhu
Assistant Professor in Information Technology
Directorate of Distance Education
Alagappa University,
Karaikudi-630003

OBJECT ORIENTED ANALYSIS AND DESIGN

Syllabi – Book Mapping Table

CONTENT

CONTENTS

BLOCK 1: INTRODUCTION

UNIT I: Object Oriented System Development

Structure

1.0 INTRODUCTION

Software development is dynamic and always undergoing major change. Today a vast number of tools and methodologies are available for system development. System development refers to all activities that go into producing information system solution. System development activities consist of system analysis, modeling, design, implementation, testing and maintenance. A software development methodology is series of processes that, if followed, can lead to the development of an application. The original goal based on the system requirements. Further we study about the unified approach, which is the methodology used for learning about object oriented system development.

1.1 OBJECTIVES

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

- Understand object oriented system development
- Describe the object basics
- Understand the evolution and foundation of object model
- Describe the elements of object model

1.2 ORTHOGONAL VIEW OF THE SOFTWARE

A software system is a set of mechanism for performing certain action on certain data

Algorithm + Data structure = Program

1.3 OBJECT ORIENTED SYSTEM DEVELOPMENT METHODOLOGY

OO development offers a different model from the traditional software development approach. This is based on functions and procedures. To develop s/w by building self contained modules or objects that can be easily replaced, modified and reused.

In OO environment, s/w is a collection of discrete object that encapsulate their data as well as the functionality to model real world objects. Each object has attributes (data) and method (function). Objects are grouped in to classes and objects are responsible for it. A chart object is responsible for things like maintaining its data and labels and even for drawing itself.

Benefits of Object Orientation

Faster development,
Reusability,
Increased quality

Object oriented technology emphasizes modeling the real world and provides us with the stronger equivalence of the real world‘s entities (objects) than other methodologies. Raising the level of abstraction to the point where application can be implemented in the same terms as they are described.

Why object orientation?

To create sets of objects that work together concurrently to produce s/w that better, model their problem domain that similarly system produced by traditional techniques.

It adapts to

1. Changing requirements
2. Easier to maintain
3. More robust
4. Promote greater design
5. Code reuse

Reasons for why object orientation works

1. Higher level of abstraction
2. Seamless transition among different phases of software development
3. Encouragement of good programming techniques
4. Promotion of reusability

Overview of the Unified Approach

The unified approach (UA) is a methodology for software development. The UA, based on methodologies by Booch, Rumbaugh, Jacobson, and others, tries to combine the best practices, processes, and guidelines. UA based on methodologies by Booch, Rumbaugh and Jacobson tries to combine the best practices, processes and guidelines along with the object management groups in unified modelling language. UML is a set of notations and conventions used to describe and model an application. UA utilizes the unified modeling language (UML) which is a set of notations and conventions used to describe and model an application.

1.4 OBJECT BASICS

Object is a combination of logic and data which represents real world entity. In an object-oriented system, all are objects. Such as numbers, arrays, records, fields, files, forms, etc.

An Object is represented as anything that can be real or abstract, in which we store data and we adopt methods through which the data is manipulated. All objects are responsible for themselves.

Example: a) A window object is responsible for things like opening, sizing, and closing itself.

b) A chart object is responsible for things like maintaining its data and labels, and even for drawing itself.

1.4.1 Objects are grouped into classes

Classes distinguish objects from one another. A class is a set of objects that share a common structure and common behaviour. A single object is called an instance of a class. A class can be a specification of variables, methods and inheritance of objects. The objective of a class is to define the properties and procedures of its objects.

Example: Class: car, property: colour.

1.4.2 Attributes: Object state and properties

The state of an object is represented by Properties. Each property can be defined in different ways in a programming language. Example: For a colour, it can be represented as text, or through paint, or a video, or an image etc.

1.4.3 Object behaviour and methods

Behaviour represents the collection of methods. It explains what the object is capable of doing. Behaviour of an particular object is described by unique procedures. There is no need of writing complex code or conditions for deciding a function in an object oriented system because an object takes responsibility of its own behaviour.

1.4.4 Objects respond to messages

The capability of an object is determined by the methods defined for it. Example draw method will tell a chart how to draw itself. For carrying out the methods a message is sent to an object and in response to that the objects perform operations. Messages are nonspecific function calls.

1.4.5 Encapsulation and information hiding

The internal data and procedures of an object are covered or hided and interfaces are provided to each object to reveal the necessary inner workings is called information hiding.

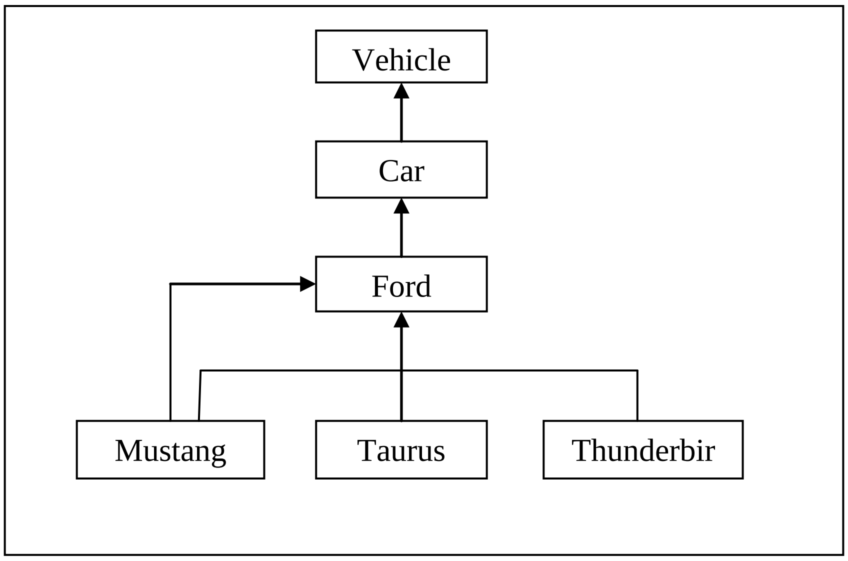
For example in C++ encapsulation is provided by public, private and protected members. The data and program of an object are encapsulated. So the user cannot see the internal side of an object instead can use the object by calling its methods.

1.4.6 Class Hierarchy

An object-oriented system organizes classes into subclass-super hierarchy. At the top of the hierarchy are the most general classes. A subclass inherits all of the properties and methods (procedures) defined in its super class. Subclasses can filter the state and behaviour inherited from its superclass.

1.4.7 Inheritance

Inheritance is a relationship between classes where one class is the parent class of another (derived) class. Inheritance allows classes to share and reuse behaviours and attributes. The real advantage of inheritance is that we can build upon what we already have and, reuse what we already have.



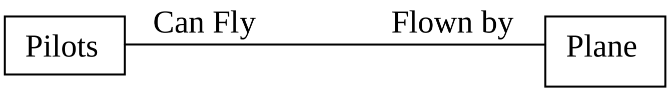
Inheritance allows reusability

1.4.8 Polymorphism

Poly gives the meaning as “Many” and morph means “form”. In object oriented systems, an object can take many different forms. Same operation can behave in a different manner in different classes is called Polymorphism. Code reusability is done more easily here.

1.4.9 Object Relationships and Associations

The relationship between classes and objects are represented as Association. They are bidirectional. Cardinality is an important issue in association where the number of instances of one class may relate to single instance of an associated class. It is described as “one” or “many”.



1.4.10 Payroll program

Consider a payroll program that processes employee records at a small manufacturing firm. This company has three types of employees:

Managers: Receive a regular salary.

Office Workers: Receive an hourly wage and are eligible for overtime after 40 hours. Production Workers: Are paid according to a piece rate.

1.4.11 Structured Approach

FOR EVERY EMPLOYEE DO

BEGIN

IF employee = manager THEN CALL computeManagerSalary

IF employee = office worker THEN CALL computeOfficeWorkerSalary

IF employee = production worker THEN CALL

```
computeProductionWorkerSalary
    END
```

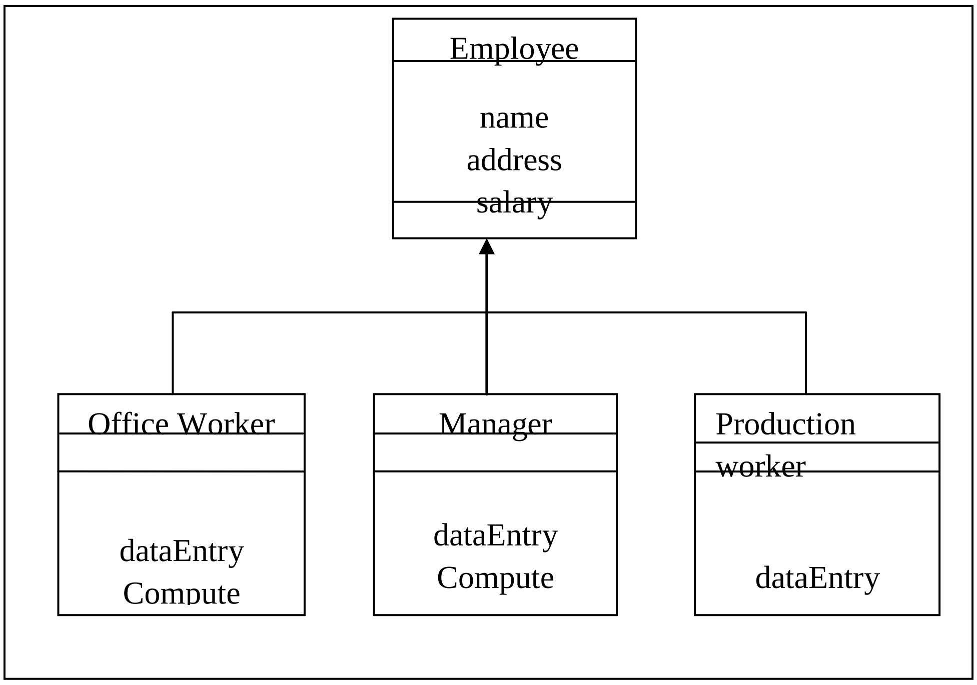
What if we add two new types of employees?

Temporary office workers are ineligible for overtime, junior production workers who receive an hourly wage plus a lower piece rate.

```
    FOR EVERY EMPLOYEE DO
    BEGIN
    IF employee = manager THEN CALL computeManagerSalary
    IF employee = office worker THEN CALL computeOfficeWorker_salary
    IF employee = production worker THEN CALL computeProductionWorker_salary
    IF employee=temporary office worker THEN CALL
                  computeTemporaryOfficeWorkerSalary
    IF employee = junior production worker THEN CALL
                          computeJuniorProductionWorkerSalary
    END
```

1.4.12 An Object-oriented Approach

The goal of OO analysis is to identify objects and classes that support the problem domain and system's requirements. Some general candidate classes are: Persons Places Things Class Hierarchy Identify class hierarchy Identify commonality among the classes Draw the general-specific class hierarchy.



Class Hierarchy for payroll application

1.4.13 Objects and Persistence

Objects have lifetime. An object can persist beyond application session boundaries, during which the object is stored in a file or a database, in some file or in a database form.

1.4.14 Meta-Classes

Everything is an object. A class is also an object. So, if it is an object, it must belong to a class. Class belongs to a class called a Meta-Class or a class' class. Meta-class is used by the compiler. For example, the meta-classes handle messages to classes, such as constructors and "new". Rather than treat data and procedures separately, object-oriented programming packages them into "objects." O-O system provides us with the set of objects that closely reflects the underlying application. Advantages of object-oriented programming are:

- The ability to reuse code,
- Develop more maintainable systems in a shorter amount of time.
- More resilient to change, and
- More reliable, since they are built from completely tested and debugged classes.

1.5 OBJECT MODEL

The elements of Object oriented technology are collectively known as object model. It covers the principles of abstraction, encapsulation, modularity, hierarchy, typing, concurrency and persistence.

Check Your Progress

1. Define object oriented technology.
2. Define UML.
3. What is an object?
4. Define Inheritance.
5. Give the advantages of object oriented programming.

1.6 THE EVOLUTION OF OBJECT MODEL

Object oriented development was founded from the best things from previous technologies. The development of more programming languages has brought these advances.

1.6.1 The Generations of Programming Languages

First generation languages (1954-1958): FORTRAN I, ALGOL 60, Flowmatic, IPL V.

These were all based on mathematical expressions.

Second generation languages (1959-1961): FORTRAN II, ALGOL

60, COBOL, Lisp.

They were based on algorithmic abstractions

Third generation languages (1962-1970): PL/1, ALGOL 68, Pascal, Simula.

They supported Data abstraction.

Generation gap (1970-1980): C, FORTRAN 77. Programming languages emerged.

Larger programs were written.

Object oriented boom (1980-1990): Smalltalk 80, C++, Ada 83, Eiffel.

Pure object oriented language emerged.

Emergence of Frameworks (1990-today): Visual Basic, Java, Python, J2EE, .NET, Visual C#, Visual Basic .NET. They provided enormous support to programmers by offering components and services which simplifies many complex tasks.

1.6.2 Topology of First and Early Second generation PL

Flat physical structures were shown which consisted only of global data and subprograms. While designing we can logically separate different kinds of data, but there is no programming language support for design decisions. These languages contain more amount of cross-coupling among subprograms, meanings of data and twisted flow of control etc. So the reliability of the system is reduced.

1.6.3 Topology of Late second and early Third generation PL

The concept of Procedural abstraction, parameter passing was supported. Nesting of subprograms, development in control structures and scope and visibility of declarations was developed. But large programming and data design was not supported.

1.6.4 Topology of late Third generation PL

To build large programs modular structure was introduced. But there was no support to check for semantics among module interfaces. There was no support for data abstraction and error could be detected only during execution.

1.6.5 Topology of Object based and Object oriented Programming Languages

Data driven design emerged. Theories for concept of a type appeared.

1.7 FOUNDATIONS OF OBJECT MODEL

1.7.1 Structured design methods

Structured design methods evolved for developers building complex systems. Algorithms were used as fundamental building blocks. It uses procedural programming language.

1.7.2 Object oriented design methods

It used classes and objects as basic building blocks. It used object based and object oriented programming languages.

Object model includes different terminologies such as Object Oriented programming, Object oriented design and Object oriented analysis.

Object oriented programming is a method where the programs are organized as collection of objects. Each object represents an instance of some class and they are members of a hierarchy of classes.

Object oriented design is a method to encompass the object oriented decomposition and it is a notation to depict logical and physical models of the system

Object oriented analysis is a method that verifies the requirements from the vision of classes and objects in the problem domain.

1.8 ELEMENTS OF THE OBJECT MODEL

A model must have any one of these elements, to be object oriented. The four major elements are

- Abstraction
- Encapsulation
- Modularity
- Hierarchy

A model can also have these elements which are useful, but not more essential in the object model. The three minor elements are −

- Typing
- Concurrency
- Persistence

1.8.1 Abstraction

Abstraction focuses on the essential features of an element or object in OOP, ignoring its extraneous or accidental properties. The essential features are relative to the context in which the object is being used. An abstraction denotes the essential characteristics of an object that distinguish it from all other kinds of objects and thus provide crisply defined conceptual boundaries, relative to the perspective of the viewer.

Example − When a class Student is designed, the attributes enrolment_number, name, course, and address are included while characteristics like pulse_rate and size_of_shoe are eliminated, since they are irrelevant in the perspective of the educational institution.

1.8.2 Encapsulation

Encapsulation binds attributes and methods together within a class. Through encapsulation, the internal details of a class can be hidden from outside. The class has methods that provide user interfaces by which the services provided by the class may be used.

1.8.3 Modularity

Modularity decomposes a problem into a set of modules to reduce the overall complexity of the problem. Modularity is the property of a system that has been decomposed into a set of cohesive and loosely coupled modules. Modularity is linked with encapsulation. Modularity can be viewed as a way of mapping encapsulated abstractions into real, physical modules having high cohesion within the modules and their inter–module interaction or coupling is low.

1.8.4 Hierarchy

Hierarchy is the ranking or ordering of abstraction. Through hierarchy, a system can be made up of interrelated subsystems, which can have their own subsystems and so on until the smallest level components are reached. Hierarchy allows code reusability.

The two types of hierarchies in OOA are −

- **"IS–A" hierarchy** − It defines the hierarchical relationship in inheritance, whereby from a super-class, a number of subclasses may be derived which may again have subclasses and so on. For example, if we derive a class Rose from a class Flower, we can say that a rose "is–a" flower.
- **"PART–OF" hierarchy** − It defines the hierarchical relationship in aggregation by which a class may be composed of other classes. For example, a flower is composed of sepals, petals, stamens, and carpel. It can be said that a petal is a "part–of" flower.

1.8.5 Typing

A type is a characterization of a set of elements. In OOP, a class is visualized as a type having properties distinct from any other types. Typing is the enforcement of the notion that an object is an instance of a single class or type. It also enforces that objects of different types may not be generally interchanged; and can be interchanged only in a very restricted manner if absolutely required to do so.

The two types of typing are −

- **Strong Typing** − Here, the operation on an object is checked at the time of compilation, as in the programming language Eiffel.
- **Weak Typing** − Here, messages may be sent to any class. The operation is checked only at the time of execution, as in the programming language Smalltalk.

1.8.6 Concurrency

Concurrency in operating systems allows performing multiple tasks or processes simultaneously. When a single process exists in a system, it is said that there is a single thread of control. However, most systems have multiple threads, some active, some waiting for CPU, some suspended, and some terminated. Systems with multiple CPUs inherently permit concurrent threads of control; but systems running on a single CPU use appropriate algorithms to give equitable CPU time to the threads so as to enable concurrency.

In an object-oriented environment, there are active and inactive objects. The active objects have independent threads of control that can execute concurrently with threads of other objects. The active objects synchronize with one another as well as with purely sequential objects.

1.8.7 Persistence

An object occupies a memory space and exists for a particular period of time. In traditional programming, the lifespan of an object was typically the lifespan of the execution of the program that created it. In files or databases, the object lifespan is longer than the duration of the process creating the object. This property by which an object continues to exist even after its creator ceases to exist is known as persistence.

Check Your Progress

6. Define object oriented analysis.
7. What is object oriented design?
8. Define Encapsulation.
9. What is typing?
10. Define Concurrency.

1.9 ANSWERS TO CHECK YOUR PROGRESS QUESTIONS

1. Object oriented technology emphasizes modelling the real world and provides us with the stronger equivalence of the real world‘s entities (objects) than other methodologies.
2. UML is a set of notations and conventions used to describe and model an application.
3. Object is a combination of logic and data which represents real world entity.
4. Inheritance is a relationship between classes where one class is the parent class of another (derived) class. Inheritance allows classes to share and reuse behaviours and attributes.
5. The ability to reuse code, develop more maintainable systems in a shorter amount of time, more resilient to change and more reliable, since they are built from completely tested and debugged classes.
6. Object oriented analysis is a method that verifies the requirements from the vision of classes and objects in the problem domain.
7. Object oriented design is a method to encompass the object oriented decomposition and it is a notation to depict logical and physical models of the system
8. Encapsulation binds attributes and methods together within a class.

9. Typing is the enforcement of the notion that an object is an instance of a single class or type.
10. Concurrency in operating systems allows performing multiple tasks or processes simultaneously.

1.10 SUMMARY

- A class describes a collection of similar objects. It is a template where certain basic characteristics of a set of objects are defined.
- The notation for an object is the same in basic form as that for a class.
- Objects usually appear as components of a larger program or a system.
- Object is an instance of a class.
- Objects have lifetime.
- The main advantage of the object-oriented approach is the ability to reuse code and develop more maintainable systems in a shorter amount of time.
- Through the interaction of these objects, functionality of systems is achieved.
- Links and associations are the basic means used for establishing relationships among objects and classes of the system.
- Association and classes are similar in the sense that classes describe objects, and association describes links.
- Multiplicity in an association specifies how many objects participate in a relationship. Multiplicity decides the number of related objects.
- 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 super classes.
- Generalization is used to refer to the relationship among classes, and inheritance is used for sharing attributes and operations using the generalization relationship.

1.11 KEYWORDS

Association

Attributes

Generalization

Inheritance

Link

Object

1.12 REVIEW QUESTIONS

1. What is system development methodology?

2. What is object oriented system development methodology?
3. What are the advantages of object-oriented development?
4. Describe the components of the unified approach.
5. What is a class? Make distinction between attributes and operations.
6. Describe the differences between the notations of class and object respectively.
7. Illustrate the concept of objects and classes with examples.
8. What is polymorphism?
9. Explain the concept of communication by message passing. Also discuss the advantages of message passing.
10. How are classes organized in an object-oriented environment?
11. What is a consumer producer relationship?
12. Elucidate the concept of link and association with example.
13. What is a formal class?
14. What is an instance?
15. Give the orthogonal views of software.

1.13 FURTHER READINGS

1. Grady Booch, Robert A.Maksimchuk et.al,Object oriented analysis and design with Applications, Pearson Education, 3rd Edition, 2009.

2. Object-oriented Software Engineering, TMH Rumbaugh, J. (2007),

3. Object-oriented Modelling and Design with UML, Pearson Education Satzinger, (2007),

UNIT II: CLASSES AND OBJECTS

NOTES

Structure

2.0 INTRODUCTION

When we use object-oriented methods to analyze or design a complex software system, basic building blocks are classes and objects. In this unit we have a detailed a study of the nature of the classes, objects and their relationships.

2.1 OBJECTIVES

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

- Understand the nature of the object & class
- Understand the relationship among objects and classes

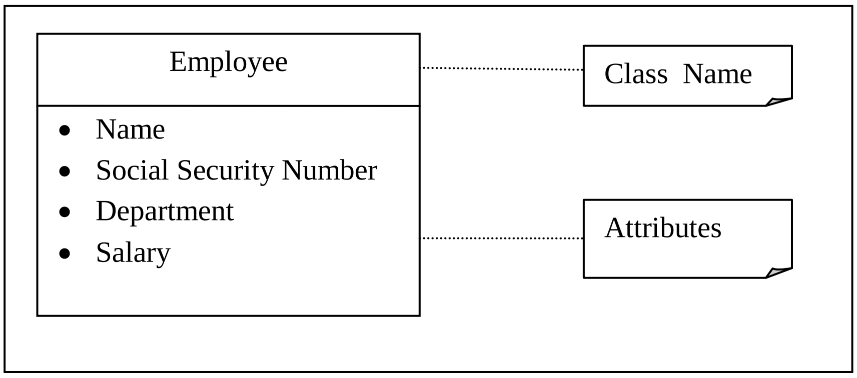
2.2 NATURE OF AN OBJECT

An object has state, behavior, and identity. The structure and behavior of similar objects are defined in their common class.

2.2.1 State

The state of an object includes all the properties of the object and the current values of each of these properties. For example, consider a vending machine that offers soft drinks. The normal behavior of these objects is that when someone inserts money in a slot and pushes a button to make a selection, a drink emerges from the machine. If a user first makes a selection and then puts money in the slot then those vending machines just sit and do nothing because the user has violated the basic assumptions of their operation. In another way, the vending machine was waiting for money first but the user ignored that and made a selection first. Also suppose that the user ignores the warning light that says, "Correct change only," and puts in extra money the vending machine does nothing. Most machines are user-hostile; they will take in the excess money.

A property is an intrinsic characteristic that makes an object unique. Properties are always static. All properties have some value which is a simple quantity. All objects within a system encapsulate some state.



Employee class with Attributes

2.2.2 Behaviour

Behaviour is how an object will act and react, to its changeable state of object affect its behaviour. In vending machine, if we don't deposit change sufficient for our selection, then the machine will probably do nothing. So behaviour of an object is a function of its state as well as the operation performed upon it. The state of an object represents the combined results of its behaviour. An operation is some action that one object performs on another in order to bring out a reaction. For example, a client might initiate the operations append and pop to grow and shrink a queue object, respectively. A client might also invoke the operation length, which returns a value denoting the size of the queue object but does not alter the state of the queue itself. Message passing is one part of the equation that defines the behaviour of an object; our definition for behaviour also notes that the state of an object affects its behaviour as well.

2.2.3 Operations

An operation is a service that a class provides to its clients. A client performs five kinds of operations upon an object. The three most common kinds of operations are the following:

- Modifier: An operation that alters the state of an object
- Selector: An operation that accesses the state of an object but does not alter the state.
- Iterator: An operation that permits all parts of an object to be accessed in some well defined order.

Two other kinds of operations that represent the infrastructure necessary to create and destroy instances of a class are

- Constructor: An operation that creates an object and/or initializes its state.
- Destructor: An operation that frees the state of an object and/or destroys the object itself.

2.2.4 Roles and Responsibilities

A role is a mask that an object wears and defines a contract between an abstraction and its clients. Responsibilities are meant to convey a sense of the purpose of an object and its place in the system. The responsibilities of an object are all the services it provides for all of the contracts it supports. The state and behaviour of an object collectively define the roles that an object may play in the world, which in turn fulfil the abstraction's responsibilities. Examples:

1. A bank account may have the role of a monetary asset to which the account owner may deposit or withdraw money. However, to a taxing authority, the account may play the role of an entity whose dividends must be reported on annually.
2. To a trader, a share of stock represents an entity with value that may be bought or sold; to a lawyer, the same share denotes a legal instrument to which are attached certain rights.
3. In the course of one day, the same person may play the role of mother, doctor, gardener, and movie critic.

2.2.5 Identity

Identity is the property of an object which distinguishes it from all others.

Example: Consider a class that denotes a display item. A display item is a common abstraction in all GUI-centric systems: It represents the base class of all objects that have a visual representation on some window and so captures the structure and behaviour common to all such objects. Clients expect to be able to draw, select, and move display items, as well as query their selection state and location. Each display item has a location designated by the coordinates x and y. First declaration creates four names and 3 distinct objects in 4 diff location. Item 1 is the name of a distinct display item object and other 3 names denote a pointer to a display item objects. Item 4 is no such objects, we properly say that item 2 points to a distinct display item object, whose name we may properly refer to indirectly as * item2. The unique identity (but not necessarily the name) of each object in preserved over the lifetime of the object, even when its state is changed. Copying, Assignment, and Equality Structural sharing takes place when the identity of an object is aliased to a second name.

Check Your Progress

1. Define State.
2. Give the kinds of operations.
3. Define Identity.

2.3 RELATIONSHIPS AMONG OBJECTS

Objects contribute to the behaviour of a system by collaborating with one another. E.g. object structure of an airplane. The relationship between any two objects encompasses the assumptions that each makes about the other including what operations can be performed. There are two kinds of objects relationships are links and aggregation.

2.3.1 Links

A link denotes the specific association through which one object (the client) applies the services of another object (the supplier) or through which are object may navigate to another. A line between two object icons represents the existence of pass along this path. Messages are shown as directed lines representing the direction of message passing between two objects is typically unidirectional, may be bidirectional data flow in either direction across a link. As a participation in a link, an object may play one of three roles:

- Controller: This object can operate on other objects but is not operated on by other objects. In some contexts, the terms active object and controller are interchangeable.
- Server: This object doesn't operate on other objects; it is only operated on by other objects.
- Proxy: This object can both operate on other objects and be operated on by other objects. A proxy is usually created to represent a real-world object in the domain of the application.

Visibility

Consider two objects, A and B, with a link between the two. In order for A to send a message to object B, B must be visible to A. Four ways of visibility

- The supplier object is global to the client
- The supplier object is a programmer to some operation of the client
- The supplier object is a part of the client object.
- The supplier object is locally declared object in some operation of the client.

Synchronization

Wherever one object passes a message to another across a link, the two objects are said to be synchronized. Active objects embody their own thread of control, so we expect their semantics to be guaranteed in the presence of other active objects. When one active object has a link to a passive one, we must choose one of three approaches to synchronization.

1. Sequential: The semantics of the passive object are guaranteed only in the presence of a single active object at a time.
2. Guarded: The semantics of the passive object are guaranteed in the presence of multiple threads of control, but the active clients must collaborate to achieve mutual exclusion.
3. Concurrent: The semantics of the passive object are guaranteed in the presence of multiple threads of control, and the supplier guarantees mutual exclusion.

2.3.2 Aggregation

Links denote peer to peer or client/supplier relationships, aggregation denotes a whole/part hierarchy, with the ability to navigate from the whole (also called the aggregate) to its parts. Aggregation is a specialized kind of association. Aggregation may or may not denote physical containment. E.g. airplane is composed of wings, landing gear, and so on. This is a case of physical containment. The relationship between a shareholder and her shares is an aggregation relationship that doesn't require physical containment.

2.4 THE NATURE OF THE CLASS

A class is a set of objects that share a common structure, behaviour and semantics. A single object is simply an instance of a class. Object is a concrete entity that exists in time and space but class represents only an abstraction. An object is not a class. Objects that do not have a common structure and behaviour may not be grouped in a class.

2.4.1 Interface and Implementation

The interface of a class provides its outside view and therefore emphasizes the abstraction while hiding its structure and secrets of its behaviour. The interface primarily consists of the declarations of all the operators applicable to instance of this class, but it may also include the declaration of other classes, constants variables and exceptions as needed to complete the abstraction. The implementation of a class is its inside view, which encompasses the secrets of its behaviour. The implementation of a class consists of the class. Interface of the class is divided into following four parts.

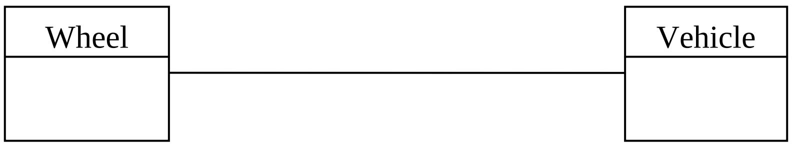
- Public: a declaration that is accessible to all clients
- Protected: a declaration that is accessible only to the class itself and its subclasses
- Private: a declaration that is accessible only to the class itself

- Package: a declaration that is accessible only by classes in the same package.

2.5 RELATIONSHIP AMONG CLASSES

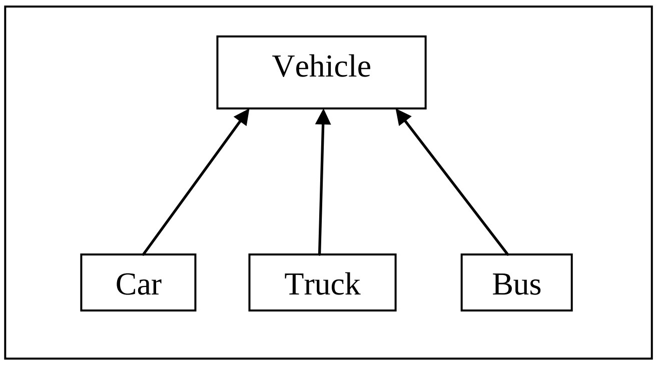
The relationships between two classes are established for one of two reasons. First, a class relationship might indicate some kind of sharing. Second, a class relationship might indicate some kind of semantic connection. There are four basic kinds of class relationships. They are Association, Aggregation, Inheritance (Generalization) and Dependency.

1. **Association:** An association is a structural relationship that describes a set of links, which is a connection among objects. The identification of associations among classes is describing how many classes/objects are taking part in the relationship. As an example for a vehicle, two of our key abstractions include the vehicle and wheels. Multiplicity denotes the cardinality of the association. There are three common kinds of multiplicity across an association are: 1. One-to-one 2. One-to-many 3. Many-to-many



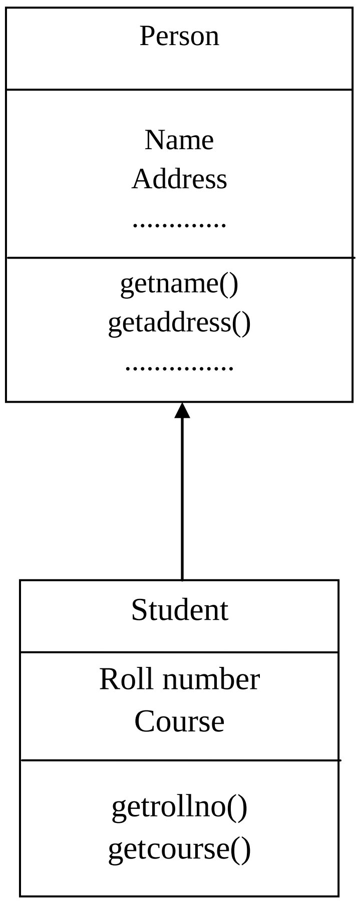
2. **Aggregation**: An aggregation is a special kind of association, representing a structural relationship between a whole and its parts. Aggregation relationships among classes have a direct parallel to aggregation relationships among the objects corresponding to these classes. Aggregation may or may not denote physical containment. E.g. airplane is composed of wings, landing gear, and so on. This is a case of physical containment.

3. **Inheritance**: Inheritance, is the most semantically interesting of the concrete relationships, exists to express generalization/specialization relationships. Inheritance is a relationship among classes wherein one class shares the structure and/or behaviour defined in one (single inheritance) or more (multiple inheritance) other classes. Inheritance means that subclasses inherit the structure of their super class.



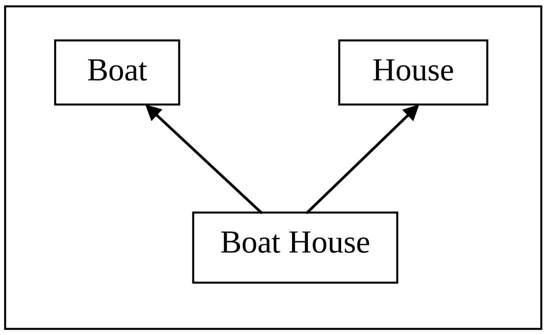
Single Inheritance: The derived class having only one base class is called single inheritance. It is a relationship among classes where in one class shares the structure and/or behaviour defined one class.

NOTES



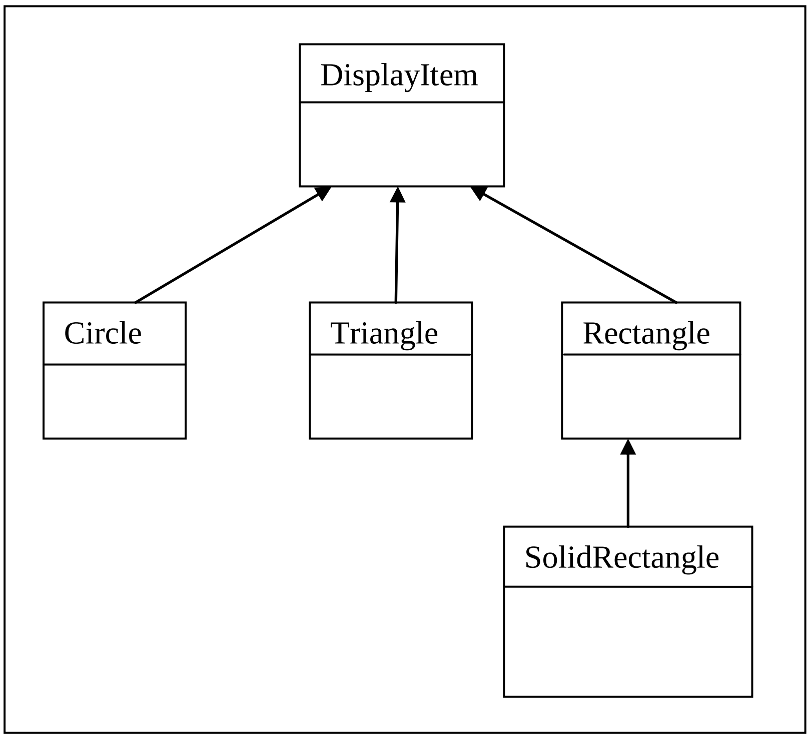
Single inheritance

Multiple Inheritance: The derived class having more than one base class is known as multiple inheritance. It is a relationship among classes where in one class shares the structure and/or behaviour defined more classes.



4. **Polymorphism**: Polymorphism is an ability to take more than one form. It is a concept in type theory wherein a name may denote instances of many different classes as long as they are related by some common super class. Any object denoted by this name is thus able to respond to some common set of operations in different ways. With polymorphism, an operation can be implemented differently by the classes in the hierarchy. Consider the class hierarchy, which has the base class DisplayItem along with three subclasses named Circle, Triangle, and

Rectangle. Rectangle also has one subclass, named SolidRectangle. In the class DisplayItem, suppose that we define the instance variable theCenter (denoting the coordinates for the center of the displayed item), along with the following operations: ■ draw: Draw the item. ■ Move: Move the item. ■ Location: Return the location of the item. The operation location is common to all subclasses and therefore need not be redefined, but we expect the operations draw and move to be redefined since only the subclasses know how to draw and move themselves.



DisplayItem Class Diagram

Check Your Progress

4. Define Links.
5. What is proxy?
6. Define Aggregation.
7. Define Interface.
8. List the four basic kinds of class relationships.

2.6 ANSWERS TO CHECK YOUR PROGRESS QUESTIONS

1. The state of an object includes all the properties of the object and the current values of each of these properties.
2. The three kinds of operations are Modifier, Selector and Operator.

3. Identity is the property of an object which distinguishes it from all others.
4. A link denotes the specific association through which one object (the client) applies the services of another object (the supplier) or through which are object may navigate to another.
5. Proxy object can both operate on other objects and be operated on by other objects. A proxy is usually created to represent a real-world object in the domain of the application.
6. Aggregation is a specialized kind of association. Aggregation may or may not denote physical containment.
7. The interface of a class provides its outside view and therefore emphasizes the abstraction while hiding its structure and secrets of its behaviour.
8. Four kinds of class relationships are Association, Aggregation, Inheritance (Generalization) and Dependency.

2.7 SUMMARY

- An object has state, behaviour and identity.
- The structure and behaviour of similar objects are defined in their common class.
- The state of an object includes static and dynamic properties.
- Behaviour is how an object acts and reacts in state changes and message passing.
- Identity is the property of an object which distinguishes it from all others.
- A class is a set of objects that shares a common structure and behaviour.
- Association, Inheritance and Aggregation are the three kinds of relationships.

2.8 KEYWORDS

- State
- Behaviour
- Constructor
- Destructor
- Identity
- Association
- Inheritance

2.9 REVIEW QUESTIONS

1. Explain the essential characteristics of an object.
2. What is a state? Explain

3. Define Interface.
4. Write about operations.
5. Distinguish between links and aggregation.
6. Explain with example the types of relationships.
7. Explain the following properties of an association by giving at least two examples of each: link attribute, role name, ordering and qualification.
8. What are different types of aggregation? Explain each with suitable examples.
9. What is inheritance? What are different uses of an inheritance? Explain.
10. Define Polymorphism.

2.10 FURTHER READINGS

Object oriented system development, Ali Bahrami, Tata McGraw Hill Edition, 2008

James Rumbaugh et.al, Object Oriented Modeling and Design, Addison Wesley, 2006

UNIT III: CLASSES AND OBJECTS

Structure

3.0 INTRODUCTION

The software development process that consists of analysis, design, implementation, testing and refinement is to transform user's needs into a software solution that satisfies those needs. The dynamics of software development provide room for shortcuts and bypasses. In this way, we need to spend more time in gathering requirements, developing requirements model and an analysis model, and then turning them into the design model. Then we can develop code quickly. A prototype is needed to be constructed for the key system components, after the products are selected. The prototype also gives users a chance to comment on the usability and usefulness of the design and let us assess the fit between the software tools selected, the functional specification and the user's needs. The man point of this unit is the idea of building software by placing emphasis on the analysis and design aspects of the software life cycle. This is intended to promote the building of high quality software.

3.1 OBJECTIVES

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

- Describes to build quality classes and objects
- Understand the system development life cycle

3.2 BUILDING QUALITY CLASSES AND OBJECTS

Object oriented analysis, design, and implementation are an iterative process. It is usually impossible to fully and correctly design the classes of an OO system at the outset of a project.

Booch proposes five metrics to measure the quality of classes:

- Coupling
- Cohesion
- Sufficiency
- Completeness
- Primitiveness

Coupling

Inheritance makes for strong coupling (generally a bad thing) but takes advantage of the re-use of an abstraction (generally a good thing).

Cohesion

Cohesion measures the degree of connectivity among the elements of a single module. Coincidental and functional cohesion are used in a desirable form.

Sufficiency

It ensures whether the class captures enough of the details to permit meaningful and efficient interaction.

Completeness

It means that the interface of the class captures all the meaningful characteristics of the abstraction. It provides an interface that is commonly usable to any client.

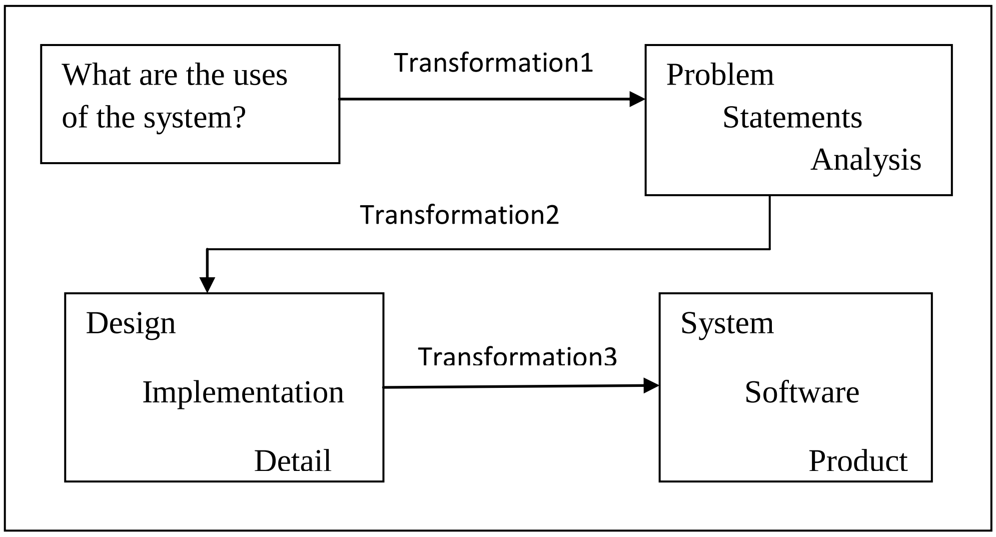
Primitive

It specifies operations that can be efficiently used only if access is given to the abstraction. Primitive classes are smaller, easier to understand, with less coupling between methods, and are more likely to be reused. Sometimes issues of efficiency or interface ease-of-use will suggest you violate the general recommendation of making a class primitive. For example, you might provide a general method with many arguments to cover all possible uses, and a simplified method without arguments for the common case.

3.3 SYSTEM DEVELOPMENT LIFE CYCLE

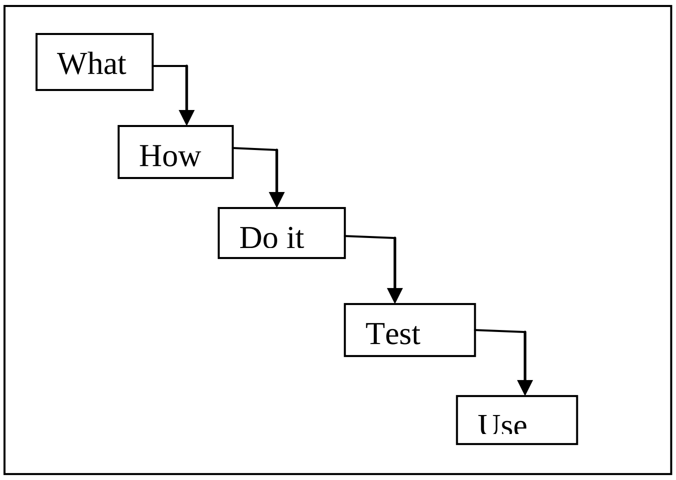
3.3.1 Software Process

The essence of the software process is the transformation of users needs to the application domain and into a software solution.


What are the uses of the system?
Transformation1
Problem Statements Analysis
Transformation2
Design Implementation Detail
Transformation3
System Software Product

Software Process

An example of software development process is the waterfall approach. It starts by deciding what is to be done. Then it decides how to accomplish them. Followed by in which we can do it. Then test the result. Finally the prepared software is used.



Waterfall software development process

3.3.2 Software Quality

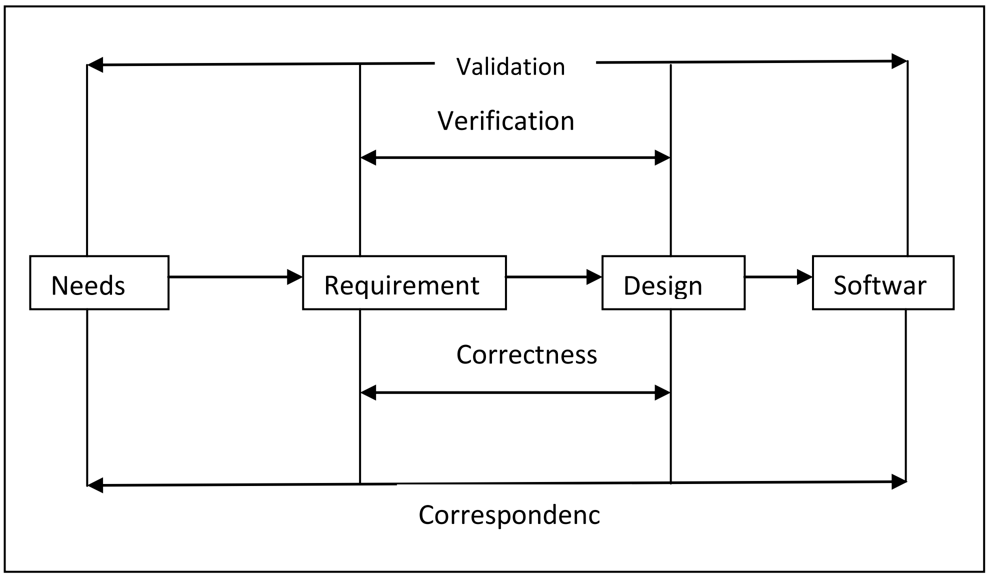
There are two basic approaches to systems testing. We can test a system according to how it has been built. Alternatively, we can test the system with respect to what it should do.

Quality Measures

Systems can be evaluated in terms of four quality measures:

- Correspondence
- Correctness
- Verification
- Validation

- Correspondence measures how well the delivered system corresponds to the needs of the operational environment. It cannot be determined until the system is in place.
- Correctness measures the consistency of the product requirements with respect to the design specification.
- Verification is to predict the correctness.
- Validation is to predict the correspondence



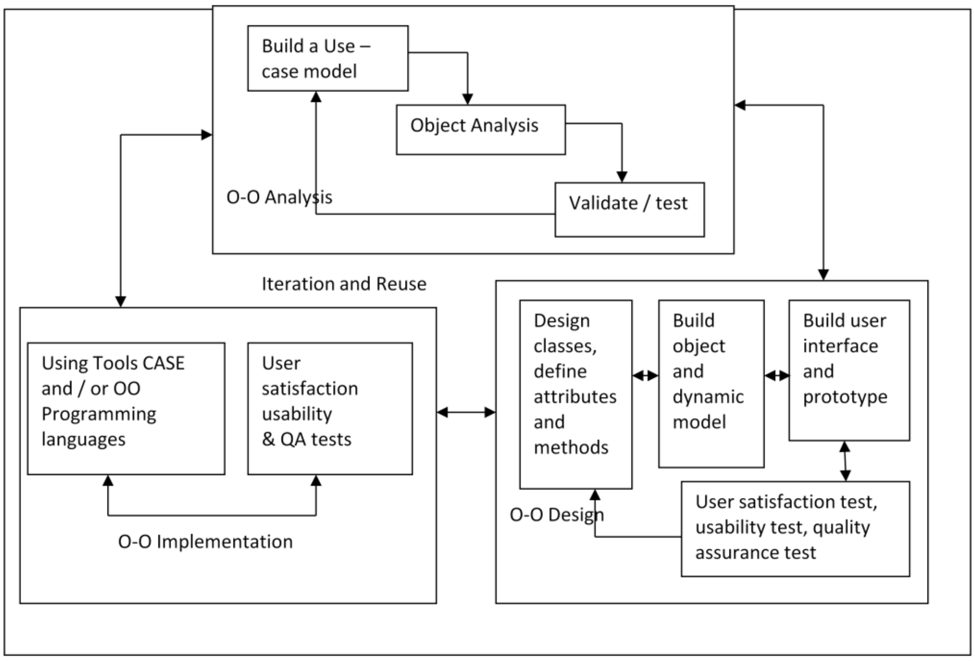
Four Quality measures for software evaluation

3.3.3 Object-Oriented Systems Development activities

The object oriented software development life cycle consists of three macro processes:

- Object Oriented Analysis
- Object Oriented Design
- Object Oriented Implementation

Based on user requirements the design decisions can be tracked.

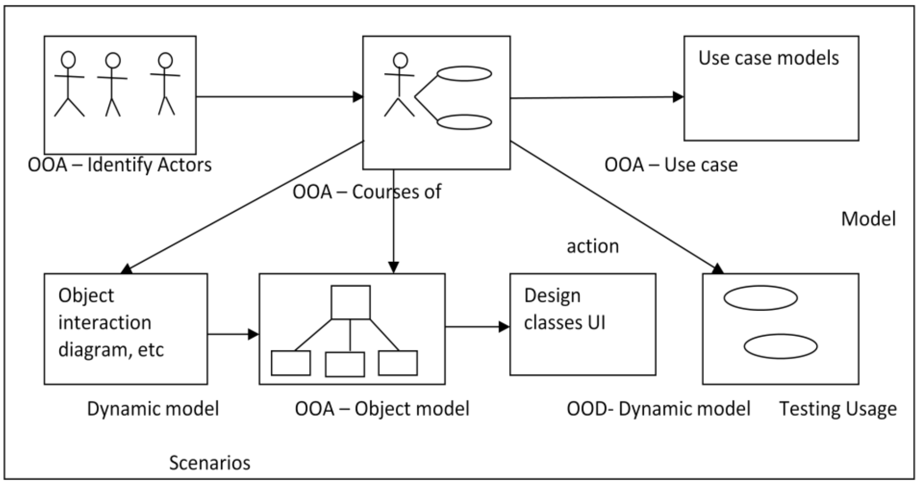

Build a Use – case model
Object Analysis
Validate / test
O-O Analysis
Iteration and Reuse
Using Tools CASE and / or OO Programming languages
User satisfaction usability & QA tests
O-O Implementation
Design classes, define attributes and methods
Build object and dynamic model
Build user interface and prototype
User satisfaction test, usability test, quality assurance test
O-O Design

Object oriented system development includes the following activities

- Object-oriented analysis.
- Object-oriented design.
- Prototyping.
- Component-based development.
- Incremental testing.

3.3.4 Use-case driven systems development

Use Case, is a name for a scenario to describe the user – computer system interaction.



3.3.5 Object-Oriented Analysis

The object-oriented analysis phase of software development is concerned with determining the system requirements and identifying classes and their relationship to other classes in the problem domain. To understand the system requirements, we need to identify the users or the actors. The intersection among objects' roles to achieve a given goal is called collaboration.

3.3.6 Object-Oriented Design

The goal of Object-Oriented design (OOD) is to design the classes identified during the analysis phase and the user interface. During this phase, we identify and define additional objects and classes that support implementation of the requirements.

First, build the object model based on objects and their relationships, then iterate and refine the model.

OOD activities include:

- Design and refine classes.
- Design and refine attributes.
- Design and refine methods.
- Design and refine structures.
- Design and refine associations.
- Design User Interface or View layer classes.
- Design data Access Layer classes.

Check Your Progress

1. What are the metrics to measure quality of classes?
2. What is the goal of Object oriented design?
3. Define Use-case.

3.3.7 Prototyping

- A Prototype enables to fully understand how easy or difficult it will be to implement some of the features of the system.
- It can also give users a chance to comment on the usability and usefulness of the design.

Types of Prototypes

- A horizontal prototype is a simulation of the interface.
- A vertical prototype is a subset of the system features with complete functionality.
- An analysis prototype is an aid for exploring the problem domain.
- A domain prototype is an aid for the incremental development of the ultimate software solution.

The purpose of this review is threefold:

1. To demonstrate that the prototype has been developed according to the specification and that the final specification is appropriate.

2. To collect information about errors or other problems in the system, such as user interface problems that need to be addressed in the intermediate prototype stage.
3. To give management and everyone connected with the project the first (or it could be second or third)

3.3.8 Component-based development (CBD)

- CBD is an industrialized approach to the software development process.
- Application development moves from custom development to assembly of pre-built,pre-tested, reusable software components that operate with each other.

3.3.9 Rapid Application Development (RAD)

- RAD is a set of tools and techniques that can be used to build an application faster than typically possible with traditional methods.
- RAD does not replace SDLC but complements it, since it focuses more on process description and can be combined perfectly with the object-oriented approach.

3.3.10 Incremental Testing

- Software development and all of its activities including testing are an iterative process.
- If you wait until after development to test an application for bugs and performance, you could be wasting thousands of dollars and hours of time.

3.3.11 Reusability

A major benefit of object-oriented systems development is reusability, and this is the most difficult promise to deliver on.

Reuse strategy

- Information hiding (encapsulation).
- Conformance to naming standards.
- Creation and administration of an object repository.
- Encouragement by strategic management of reuse as opposed to constant development.
- Establishing targets for a percentage of the objects in the project to be reused (i.e.,50 percent reuse of objects).The essence of the software process is the transformation of users‘ needs into a software solution. The O-O SDLC is an iterative process and is divided into analysis, design, prototyping/ implementation, and testing.

Check Your Progress

4. Give the purpose of prototyping.
5. Expand and define CBD.
6. Define RAD.

3.4 ANSWERS TO CHECK YOUR PROGRESS QUESTIONS

1. Coupling, Cohesion, Sufficiency, Completeness, Primitiveness
2. The goal of Object-Oriented design (OOD) is to design the classes identified during the analysis phase and the user interface.
3. Use Case, is a name for a scenario to describe the user – computer system interaction.
4. A Prototype enables to fully understand how easy or difficult it will be to implement some of the features of the system.
5. Component Based Development. Application development moves from custom development to assembly of pre-built, pre-tested, reusable software components that operate with each other.
6. RAD is a set of tools and techniques that can be used to build an application faster than typically possible with traditional methods.

3.5 SUMMARY

- The essence of the software process is the transformation of users' needs into a software solution.
- For high quality of software four quality measures are described: Correspondence, correctness, verification and validation.
- Object oriented system development consists of three macro processes: object oriented analysis, object oriented design and object oriented implementation.
- Object oriented analysis builds a use case model and interaction diagrams to identify users needs.
- Object oriented design focuses on design classes and their protocol, building class diagrams, user interfaces and prototypes, testing user satisfaction and usability.
- Software components are functional units or building blocks offering a collection of reusable services.

3.6 KEYWORDS

- Use case
- Object Oriented Analysis
- Prototype
- Rapid Application Development

3.7 REVIEW QUESTIONS

1. Give the metrics to measure the quality of classes.
2. Define prototypes.

3. What is Reusability?
4. Describe the OOD activities.
5. Explain SDLC with neat diagram.

3.8 FURTHER READINGS

1. Object oriented system development, Ali Bahrami, Tata McGraw Hill Edition
2. James Rumbaugh et.al, Object Oriented Modeling and Design, Addison Wesley
3. Grady Booch, Robert A.Maksimchuk et.al,Object oriented analysis and design with Applications, Pearson Education, 3rd Edition

BLOCK 2: OBJECT ORIENTED METHODOLOGIES

UNIT IV: Methodologies

Structure

4.0 INTRODUCTION

Many methodologies are available to choose from for the system development. Each methodology is based on modeling the business problem and implementation in an object oriented fashion. The Rumbaugh et al method has a strong method for producing object models. Jacobson et al have a strong method for producing user-driven requirement and object oriented analysis model. Booch has a strong method for producing detailed object oriented design models.

NOTES

4.1 OBJECTIVES

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

- Understand various methodologies for system development
- Describe Rumbaugh OMT
- Describe Booch methodology
- Describe Jacobson et al methodology

4.2 RUMBAUGH OBJECT MODELING TECHNIQUE

It gives the dynamic behaviour of objects in a system using the OMT dynamic model.

- There are four phases available

Analysis – results are objects, dynamic and functional models.

System design – gives a structure of the basic architecture.

Object design – produces a design document.

Implementation – produces reusable code.

- OMT separates modeling in to three different parts

Object Model – presented by object model and the data dictionary.

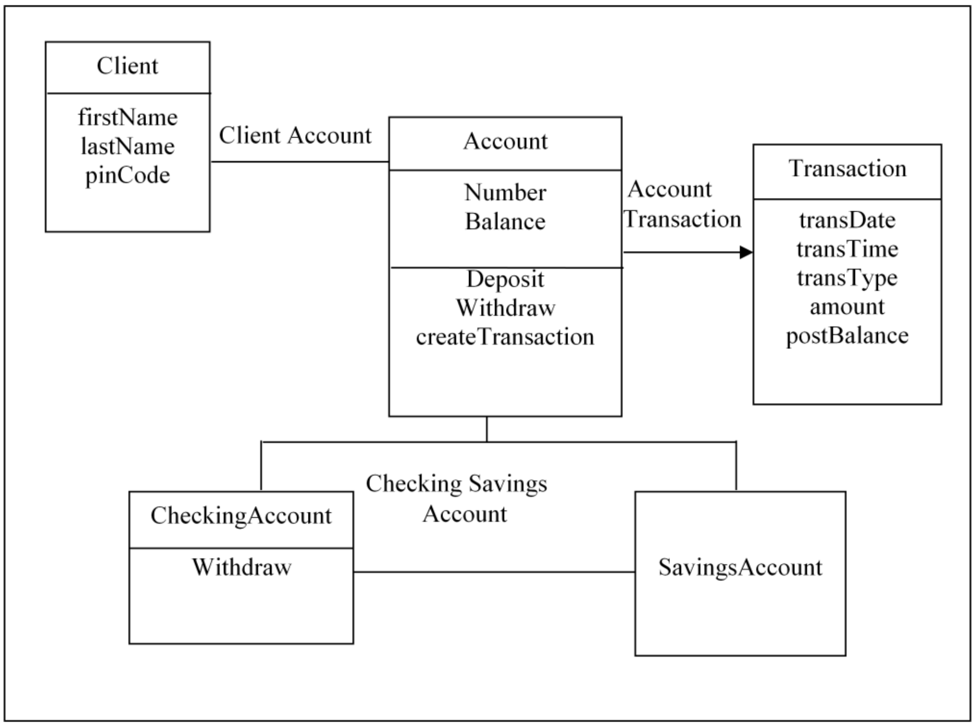
Dynamic model - presented by the state diagrams and event flow diagrams.

Functional Model – presented by data flow and constraints.

4.2.1 Object Model:-

The structure of objects in a system is described by object model and also, their identity and relationships to other objects, attributes, and operations are also defined. The object model is represented graphically with an object diagram.

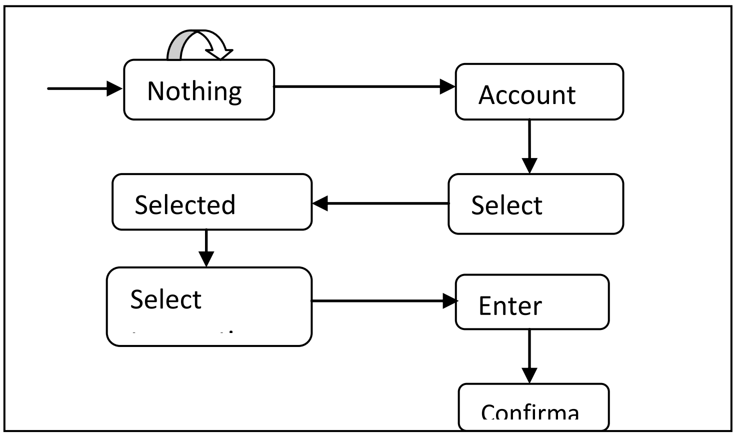
NOTES



OMT object model of a bank system. Boxes represent classes.

4.2.2 The OMT Dynamic Model:

OMT provides a detailed and comprehensive dynamic model. The OMT state transition diagram is a network of states and events. Each state receives one or more events, at which it makes the transition to the next state. The next state depends on the current state as well as the events.



State transition diagram for the bank application interface

- Round boxes represents states.
- Arrows represents transitions.

4.2.3 The OMT Functional Model

The OMT data flow diagram (DFD) shows the flow of data between different processing in a business. Data Flow Diagrams use four primary symbols:

NOTES

- The *process* is any function being performed
- The *data flow* shows the direction of data element movement
- The *data store* is a location where data are stored.
- An *external entity* is a source or destination of a data element.
- Rumbaugh OMT methodology provides one of the strong tolls set for the analysis and design of object-oriented system.

4.3 BOOCH METHODOLOGY

Booch methodology helps to design our system using the object paradigm. It is criticized for its large set of symbols. It consists of the following diagrams:

- Class diagrams
- Object diagrams
- State transition diagrams
- Module diagrams
- Process diagrams
- Interaction diagrams

There are mainly two processes available

- Macro development process
- Micro development process.

4.3.1 Macro development process

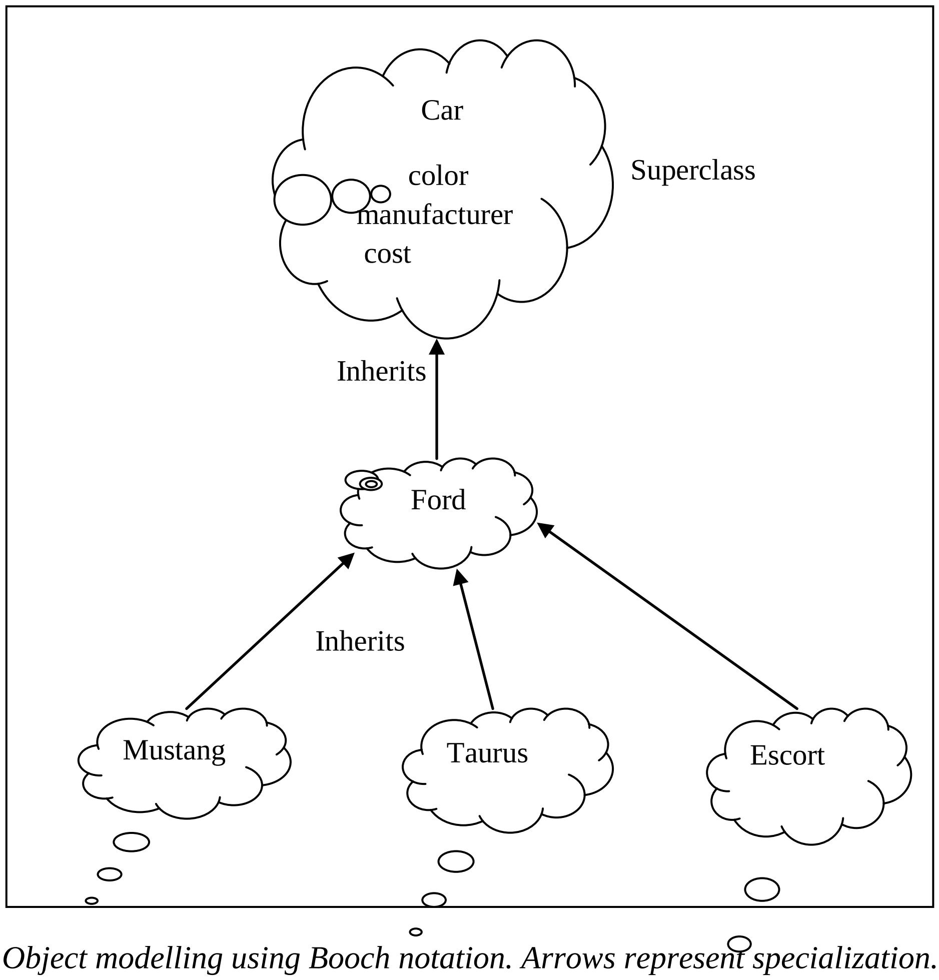
The primary concern of this process is technical management of the system.

The Steps involved here are:

Conceptualization - Establishes core requirements and develops a prototype.

Analysis and development of the model - The class diagram describes the roles and responsibilities of objects. The object diagram describes the desired behaviour of the system.

NOTES



Object modelling using Booch notation. Arrows represent specialization. Taurus is subclass of the class Ford

Design or create the system architecture - The class diagram decides what classes exist and how they relate to each other, the object diagram decides what mechanisms are used, the module diagram maps out where each class and object should be declared, and the process diagram to determine to which processor to allocate a process.

Evolution or implementation – It refines the system through iteration.

Maintenance – Can make localized changes to the system to add new requirements and eliminate bugs.

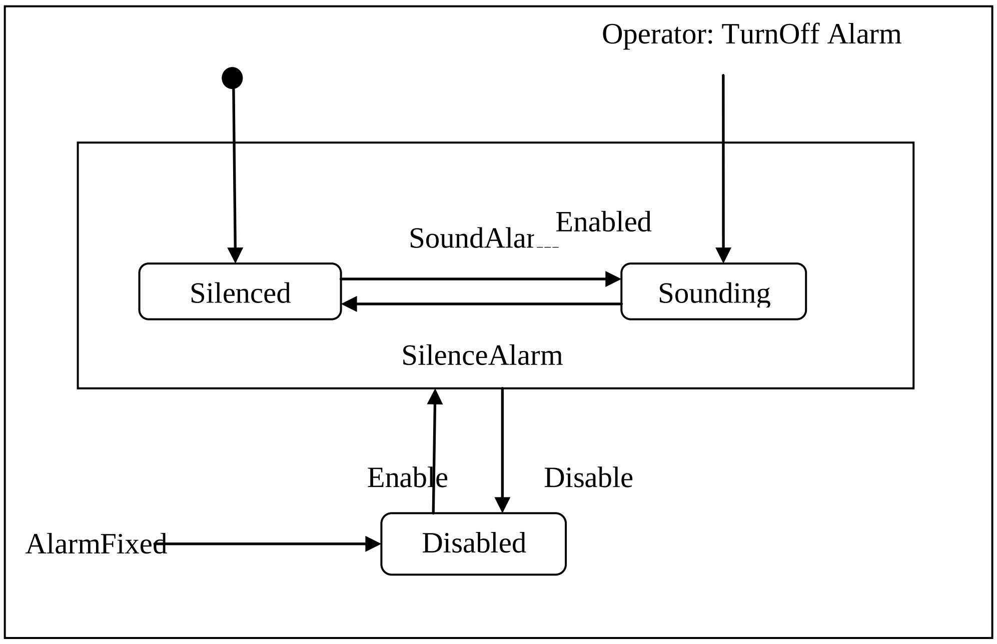
4.3.2 Micro development process

The micro development process is a description of the day-to-day activities.

Steps involved:

- Identify classes and objects
- Identify classes and object semantics
- Identify classes and object relationships
- Identify classes and object interfaces and implementation

NOTES



An alarm class state transition diagram with Booch notation

Check Your Progress

1. Define Object model.
2. What is state transition diagram?
3. Define DFD.
4. What is a class diagram?

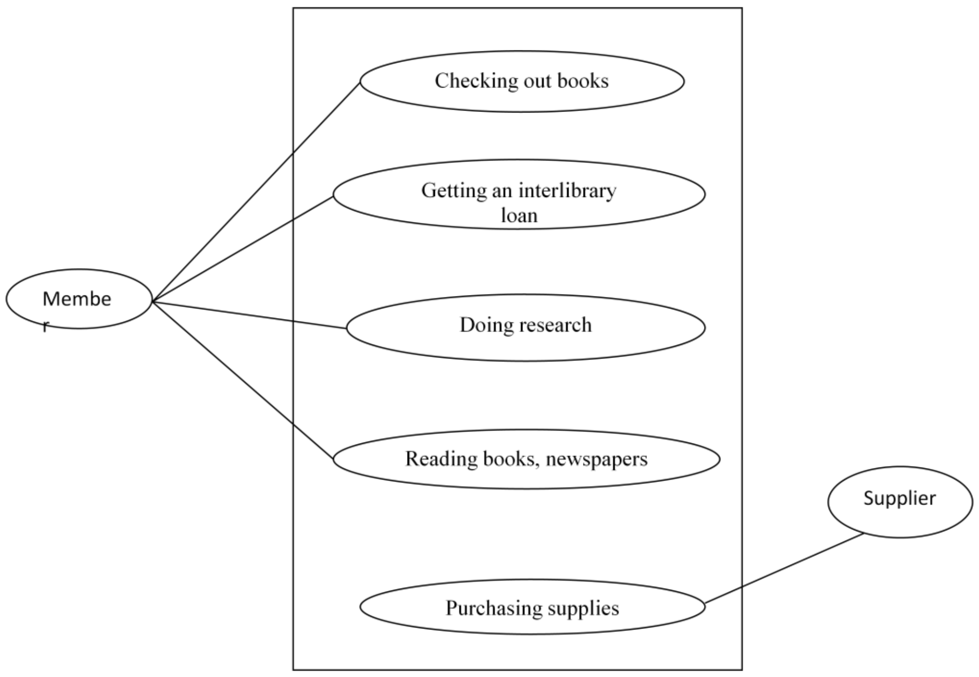
4.4 JACOBSON METHODOLOGY:

This methodology covers the entire life cycle and stress traceability between the different phases both forward & backward. It consists of:

- OOBE (O-O Business Engineering)
- OOSE (O-O Software Engineering) also called
  - OBJECTORY (Object Factory for Software Development)
- Use cases
  - Scenarios for understanding system requirements.
  - Non formal text with no clear flow of events.
  - Text easy to read.
  - Formal style using pseudo code.
  - Can be viewed as concrete or abstract (not initiated by actors).
    - Understanding system requirements
    - Interaction between user and system

NOTES

- It captures the goal of the user and responsibility of the system to its users



Library

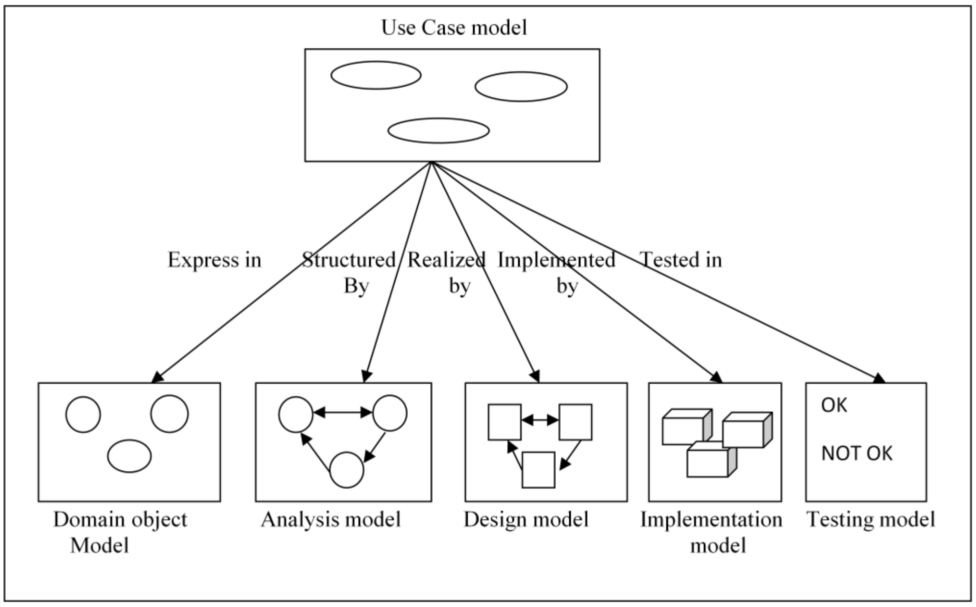
4.4.1 Object oriented software Engineering: Objectory

- OOSE is also called Objectory with the specific aim to fit the development of large, real time systems.
- Development process is also called as use case driven development.

Objectory is built around several different models such as:

- Use-case model: The use-case model defines the outside and inside of the system behaviour
- Domain Object model: The object of the real world are mapped into the domain object model.
- Analysis Object model: It presents how the source code should be carried out and written.
- Implementation model: The implementation model represents the implementation of the system.
- Test model: It includes the test plans, specifications, and reports.

NOTES



Use case model

4.4.2 Object Oriented Business Engineering

- OOBE is object modeling at the enterprise level. (Use case are also central here)
- OOBE consists of :
  - Analysis phase

The analysis phase defines the system to be built in terms of the

  - problem-domain object model
  - the requirements model
  - analysis model

  - Design & Implementation phase
  - The implementation environment must be identified for the design model.
  - Testing phase: Unit, integration & system testing.
  - This level includes unit testing, integration testing and system testing.

4.5 SHALER/MELLOR METHOD

The Shaler–Mellor method is also known as Object-Oriented Systems Analysis (OOSA) or Object-Oriented Analysis (OOA). It makes the documented analysis so precise that it is possible to implement the analysis model directly by translation to the target architecture.

Shlaer-Mellor method is based on

- separation of subject matters
- I/O, User Interaction, Application domain, Software Architecture
- specification of (executable) model for each subject matter

- translation of these models into code
- The different categories of subject matters (domains) are
- application domain (which are visible to end user)
- services domain (they are more general domains)
- software architecture domain (data, control, structures and time)
- implementation domain (OS and programming languages)

Separate teams can work on separate domains.

4.5.1 Translation

The implementation is always generated (either manually, or typically, automatically) directly from the analysis in the translative approach. In Shaler–Mellor method **virtual machine** executes any Shlaer–Mellor analysis model for any particular hardware/software platform combination.

4.5.2 Semantic decomposition

This technique proposes a semantic decomposition in multiple domains

- The split between analysis and design models: The analysis domain defines what the system must do, the design domain is a model shows how virtual machine operates for a particular hardware and software platform. These models are disjoint, the only connection being the notation used to express the models.
- Decomposition within the analysis domain: This considers how the system requirements are modelled, and grouped, around specific, disjoint, subject.

4.5.3 Precise action language

For automated code generation, the requirement is to model the actions within the finite-state machines that is used to express dynamic behaviour of Shaler–Mellor objects. Shlaer–Mellor is unique among object-oriented analysis methods in expressing such sequential behavior graphically as Action Data Flow Diagrams (ADFDs). The tools that supported Shlaer–Mellor, provided a precise action language. The action languages are written in textual form.

4.5.4 Test and simulation

The translative approach of the Shlaer–Mellor method gives itself to automated test and simulation environments. The concept of the Shlaer–Mellor virtual machine makes testing useful and productive. Shlaer–Mellor is an event-driven, message-passing environment. Shlaer–Mellor virtual machine mandates a prioritised event mechanism built around State Models, which allows for concurrent execution of actions in different state machines. Since any implementation of Shlaer–Mellor requires this model to be fully supported, testing under simulation can be a very close model of testing on target platform. The functionality heavily dependent upon timing constraints may be difficult to test, the majority of system behaviour is highly predictable due to the prioritized execution model.

4.6 COAD-YOURDON METHODOLOGY

The strength of this method is its system analysis. It is based on a technique called "SOSAS", which has five steps that build the analysis part of their methodology.

1. Subjects – It is a data flow diagrams for objects.
2. Objects – It identifies the object classes and the class hierarchies.
3. Structures – This method decompose structures into two types, classification structures and composition structures. Classification structures cope up with the inheritance connection between related classes, while composition structures carry out all of the other connections among classes.
4. Attributes
5. Services - All the behaviours or methods for each class are identified here.

In the following analysis, Coad and Yourdon define four parts that make up the design part of their methodology. The steps of system design are:

- The problem domain component - Defines the classes that should be in the problem domain.
- The human interaction component - Defines the interface classes between objects.
- The task management component - Defines where system-wide management classes are identified.
- The data management component - Identifies the classes needed for database access methods.

Check Your Progress

5. What is the aim of OOSE?
6. Give the advantage of Shaler-Mellor method.
7. Define classification structure.

4.7 ANSEWERS TO CHECK YOUR PROGRESS QUESTIONS

1. The structure of objects in a system is described by object model and also, their identity and relationships to other objects, attributes, and operations are also defined.
2. OMT state transition diagram is a network of states and events.

3. The OMT data flow diagram (DFD) shows the flow of data between different processing in a business.
4. The class diagram describes the roles and responsibilities of objects.
5. OOSE is also called Objectory with the specific aim to fit the development of large, real time systems.
6. It makes the documented analysis so precise that it is possible to implement the analysis model directly by translation to the target architecture.
7. Classification structures cope up with the inheritance connection between related classes.

4.8 SUMMARY

- Object oriented methodology is a new system development approach, encouraging and facilitating re-use of software components.
- Object oriented methodology employs international standard Unified Modeling Language (UML) from the Object Management Group (OMG).
- Object model depicts the object classes and their relationships as a class diagram, which represents the static structure of the system.
- Rumbaugh model gives the dynamic behaviour of objects in a system using the OMT dynamic model.
- The OMT state transition diagram is a network of states and events.
- The OMT data flow diagram (DFD) shows the flow of data between different processing in a business.
- The class diagram describes the roles and responsibilities of objects. The object diagram describes the desired behaviour of the system.
- OOSE is also called Objectory with the specific aim to fit the development of large, real time systems.
- Shlaer–Mellor is unique among object-oriented analysis methods in expressing such sequential behaviour graphically as Action Data Flow Diagrams.

4.9 KEYWORDS

- UML
- Object Management Group
- State transition
- Objectory

NOTES

4.10 REVIEW QUESTIONS

1. What are the phases of OMT?
2. Compare functional and dynamic model.
3. What are the diagrams used in Booch methodology?
4. What are the steps involved in Macro development process in Booch methodology?
5. Write short note on Objectory.
6. What are the various models of objectory?
7. Explain Rumbaugh's Object Modeling Technique?
8. Compare and Contrast the Booch and Jacobson methodology.
9. Write a note on Shaler–Mellor method.
10. Explain Coad and Yourdon methodology.

4.11 FURTHER READINGS

1. Booch G.,"Object Oriented Analysis And Design",Addison-Wesley Publishing Company,1994.
2. Rambaugh J, Blaha.M. Premeriani, W., Eddy F and Loresen W.,"Object Oriented Modeling and Design", PHI, 1997.

UNIT V: PATTERNS

Structure

5.0 INTRODUCTION

In systems development the process can be improved significantly if a system can be analyzed, designed and built from predefined system components. We need a body of literature to help software developers resolve commonly encountered, difficult problems and a vocabulary for communicating insight and experience about these problems and their solutions. The primary focus here is not so much on technology as on creating a culture to document and support engineering architecture and design. Patterns are being largely used for software architecture and design and for organizations, specification models and many other aspects of software development process.

5.1 OBJECTIVES

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

- Understand about patterns
- Describe pattern templates
- Understand Frameworks

5.2 PATTERNS

Pattern is instructive information that captures the essential structure and insight of a successful family of proven solutions to a recurring problem that arises within a certain context and system of forces. The main idea behind using patterns is to provide documentation to help categorize and communicate about solutions to recurring problems. The pattern has a name to facilitate discussion and the information it represents. A good pattern will do the following:

- It solves a problem. Patterns capture solutions, not just abstract principles or strategies.

- It is a proven concept. Patterns capture solutions with a track record, not theories or speculation.
- •The solution is not obvious. The best patterns generate a solution to a problem indirectly

— a necessary approach for the most difficult problems of design.

- It describes a relationship. Patterns do not just describe modules, but describe deeper system structures and mechanisms.
- The pattern has a significant human component.
- All software serves human comfort or quality of life; the best patterns explicitly appeal to aesthetics and utility.

Generative patterns – Tells us how to generate something.

- observed in a system
- descriptive and passive

Non generative patterns –

- generate systems or parts of systems
- perspective and active

5.3 PATTERNS TEMPLATES

Name:- Name allows to use a single word or short phrase to refer to the pattern and the knowledge and structure it describes. Some pattern forms also provide a classification of the pattern in addition to its name.

Problem:-A statement of the problem that describes its goals and objectives it wants to reach within the given context and forces. The forces oppose these objectives as well as each other.

Context:- The preconditions under which the problem and its solution seem to recur and for which the solution is desirable. It can be thought of as the initial configuration of the system before the pattern is applied to it.

Forces:- A description of the relevant forces and constraints and how they interact or with one another and with the goals that the user wish to achieve. A concrete scenario that serves as the motivation for the pattern frequently is employed.

Solution:- Static relationships and dynamic rules describing how to realize the desired outcome. It describes how to construct the necessary products. It encompasses the pictures, diagrams and prose that identify the pattern structure, and their participants and collaborations to show how the problem is solved.

Examples:- One or more sample applications of the pattern that illustrate a specific initial context; how the pattern is applied to and transforms that context.

Resulting context:- The state or configuration of the system after the pattern has been applied, including the consequences of applying the pattern and other problems and patterns that may arise from the new context.

Rationale:- Steps or rules in the pattern and also of the pattern as a whole in terms of how and why it resolves it forces in a particular way to be in alignment with desired goals.

Related patterns:- The static and dynamic relationships between this pattern and others within the same pattern language or system. Related pattern often share common forces. They also frequently have an initial or resulting context that is compatible with the resulting or initial context of another pattern.

Known uses:- The known occurrences of the pattern and its application within existing systems.

Check Your Progress

1. Define patterns.
2. Difference between generative & non generative patterns.
3. What is a related pattern?

5.4 ANTIPATTERNS

A pattern represents a “best practice” whereas an antipattern represents “worst practice” or a “lesson learned”. Antipattern come in two varieties:

- Those describe a bad solution to a problem that resulted in a bad situation.
- Those describing how to get out of a bad situation and how to proceed from there to a good solution.

5.5 CAPTURING PATTERNS

Guidelines for capturing patterns:

- Focus on practicability.
- Aggressive disregard of originality.
- Non anonymous review.
- Writers' workshops instead of presentations.
- Careful editing

5.6 FRAMEWORKS

Frameworks are a way of delivering application development patterns to support best practice sharing during application development. A frame work is a way of presenting a generic solution to a problem that can be applied to all levels in a development. Several design patterns in fact a framework can be viewed as the implementation of a system of design patterns.

NOTES

The major differences between design patterns and frameworks as follows

- Design patterns are more abstract than frameworks.
- Design patterns are smaller architectural elements than frameworks.
- Design patterns are less specialized than frameworks.

Check Your Progress

4. Define anti patterns.
5. What is a framework?

5.7 ANWERS TO CHECK YOUR PROGRESS QUESTIONS

1. Pattern is instructive information that captures the essential structure and insight of a successful family of proven solutions to a recurring problem that arises within a certain context and system of forces.
2. Generative patterns – Tells us how to generate something. Non-Generative patterns – generate systems or parts of systems.
3. Related pattern often share common forces. They also frequently have an initial or resulting context that is compatible with the resulting or initial context of another pattern.
4. Anti pattern represents "worst practice" or a "lesson learned".
5. A frame work is a way of presenting a generic solution to a problem that can be applied to all levels in a development.

5.8 SUMMARY

- Patterns provide documentation to help categorize and communicate about solutions to recurring problems.
- Generative patterns tell us how to generate something.
- Non generative patterns generate systems or parts of systems.
- Name allows using a single word or short phrase to refer to the pattern.
- Antipattern represents "worst practice" or a "lesson learned".
- A pattern should help its users comprehend existing systems, customize systems to fit user needs, and construct new system. The process of looking for patterns to document is called pattern mining.
- A frame work is a way of presenting a generic solution to a problem that can be applied to all levels in a development.

5.9 KEYWORDS

- Patterns
- Generative Patterns
- Non Generative Patterns
- Anti patterns
- Frameworks

5.10 REVIEW QUESTIONS

1. Differentiate between Pattern and Frameworks.
2. What are the components in pattern template?
3. Define anti-patterns.
4. Define pattern mining. Give the steps involved in capturing pattern.
5. Define frame work.
6. Describe patterns and the various pattern templates?

5.11 FURTHER READINGS

1. Booch G.,"Object Oriented Analysis And Design",Addison-Wesley Publishing Company,1994.
2. Rambaugh J, Blaha.M. Premeriani, W., Eddy F and Loresen W.,"Object Oriented Modeling and Design", PHI, 1997.

NOTES

UNIT VI: THE UNIFIED APPROACH

Structure

6.0 INTRODUCTION

The approach is based on the best practices that have proven successful in system development. The unified approach utilizes the unified modelling language to describe, model and document the software development process. The main motivation here is to combine the best practices, processes, methodologies and guidelines along with UML notations and diagram for better understanding object- oriented concepts and system development.

6.1 OBJECTIVES

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

- Describe Unified approach
- Understand about UML
- Understand about UML class diagrams
- Describe Packages
- Describe UML extensibility

6.2 UNIFIED APPROACH

It establishes a unifying and unitary framework by utilizing UML.

The processes are:

- Use case driven development.

- Object oriented analysis.
- Object oriented design.
- Incremental development and prototyping
- Continuous testing.

Methods and technologies employed include:

UML:- For modelling Unified Modelling approach is used

Layered approach

Repository:- Repository for object-oriented system development patterns and frameworks

CBD:- Component based development

The Unified Approach allows iterative development by allowing to go back and forth between the design and the modeling or analysis phase. It makes backtracking very easy and departs from the linear waterfall process, which allows no form of backtracking.

Object oriented Analysis

- Identify actors.
- Develop a simple process model.
- Develop the use case.
- Develop interaction diagrams.
- Identify classes.

Object oriented design

- Design classes, attributes, methods etc.
- Design the access layer.
- Design the prototype user interface.
- User satisfaction and usability tests.
- Iterate and refine the design.

Continuous testing

- Iterate and reiterate until satisfied with the system

UML

- Universal language for modelling systems
- Has standard notation

Repository

- Allows maximum reuse of previous experience
- Should be accessible by many people

Layered approach

1. The business layer - Displays results and provides data access details

2. The user interface or view layer –
   - Responds to user interaction: It is designed to translate actions given by the user, such as clicking on a button or selecting from a menu
   - Displaying business objects: This layer paints a best possible picture of the business objects for the user.
3. The access layer
   - Translate request: The access layer must be able to translate any data related requests from the business layer in to the appropriate protocol for data access.
   - Translate results: The access layer also must be able to translate the data retrieved back into the appropriate business objects and pass those objects back up into the business layer.

6.3 UNIFIED MODELLING LANGUAGE

This object-oriented system of notation was evolved from the work of Grady Booch, James Rumbaugh, Ivar Jacobson, and the Rational Software Corporation. These renowned computer scientists combined their respective technologies into a single, standardized model.

Now, UML is accepted by the Object Management Group (OMG) as the standard for modelling object oriented programs. UML survival introduces the Unified Modeling Language and leads us through an orderly progress towards mastery of the language.

Model:

A Model is an abstract representation of a system Constructed to understand the system prior to building or modifying it.

Why Modelling:

Building a model for a software system prior to its construction is as essential as having a blue print for building a large building. Good models are essential for communication among project teams. It has

1. Model elements - fundamental modelling concepts semantics
2. Notation - Visual rendering of model elements.
3. Guidelines - Expression of usage within the trade.

This visual notation provides benefits related to clarity, familiarity and maintenance.

Clarity: Much better at picking out errors.

Familiarity: Similar to the way in which information is actually stored.

Maintenance: Visual notation can improve the maintainability of a system.

Advantages:

1. Easier to express complex ideas
2. Reduction of complexity.
3. Cost is lower.
4. Manipulation is easier.

6.4 STATIC MODEL

It can be viewed as a snapshot of a system's parameters at rest or at a specific point in time Ex: Class diagram

6.5 DYNAMIC MODEL

It can be viewed as a collection of procedures or behaviours that taken together reflect the behaviour of a system over time ex: Interaction diagram.

Check Your Progress

1. Give the purpose of access layer.
2. Define static model.
3. Define dynamic model.

6.6 INTRODUCTION TO UML

The unified modeling language (UML) is a language for specifying, constructing, visualizing, and documenting the software system and its components.

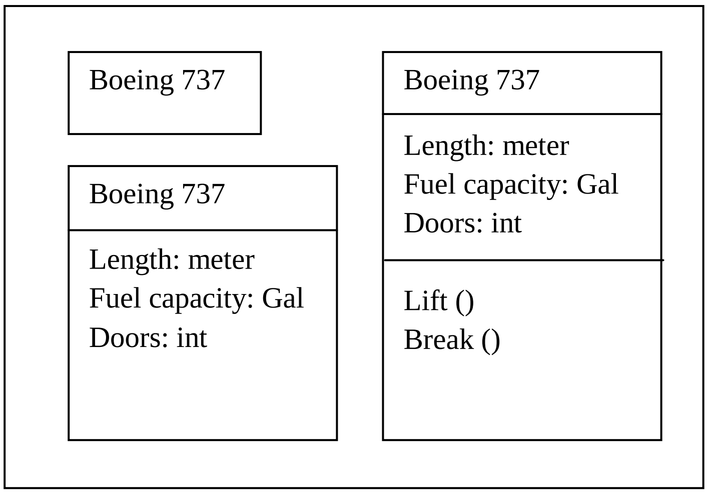
- UML is graphical language with set of rules and semantics
- The rules and semantics of a model are expressed in English, in a form of known as Object constraint language (OCL).
- The UML defines nine graphical diagrams:

1. Class diagram (static)
2. Use-case diagram
3. Behavior diagrams (dynamic):
    - 3.1. Interaction diagram:
        - •3.1.1. Sequence diagram
        - •3.1.2. Collaboration diagram
    - 3.2. Statechart diagram
    - 3.3. Activity diagram
4. Implementation diagram:
    - 4.1. Component diagram
    - 4.2. Deployment diagram

6.7 UML CLASS DIAGRAM

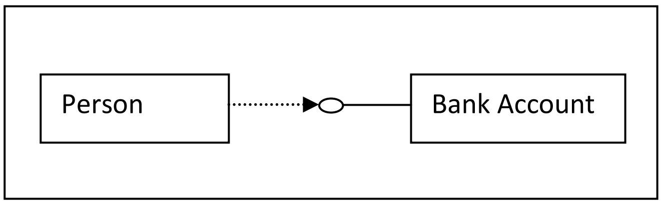
UML Class Diagram is also referred to as Object modeling. It is a collection of static modelling elements. The system is formed with objects, because in the object-oriented viewpoint, objects are the primary abstraction. Each object in the problem domain describes the structure and the relationship among objects.

Class notation – rectangle with three compartments.



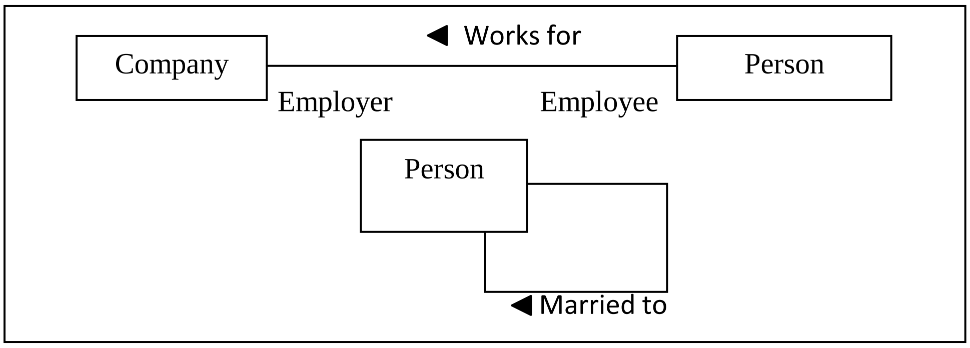
Object diagram – instance of a class diagram.

Class interface notation – small circle with the interface name connected to the class.



Interface notation of a class

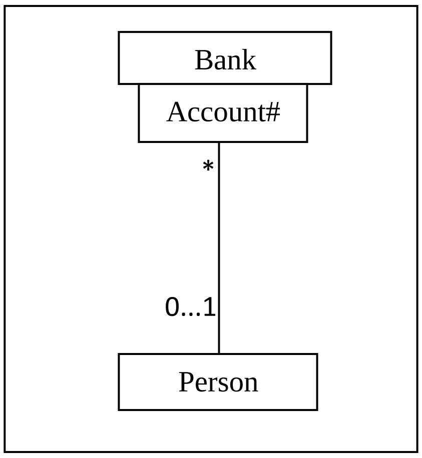
Binary association – solid path connecting two classes



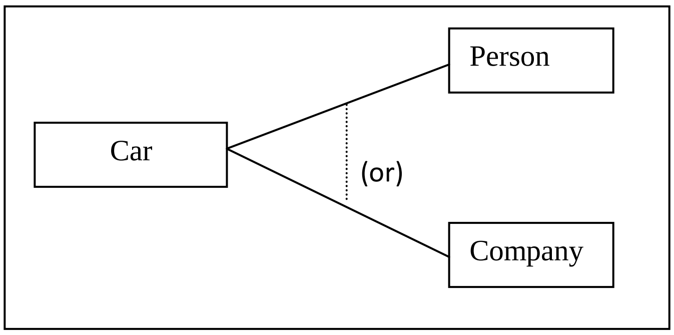
Qualifier – small rectangle attached to the end of an association.

Multiplicity – specifies the range of allowable associated classes.

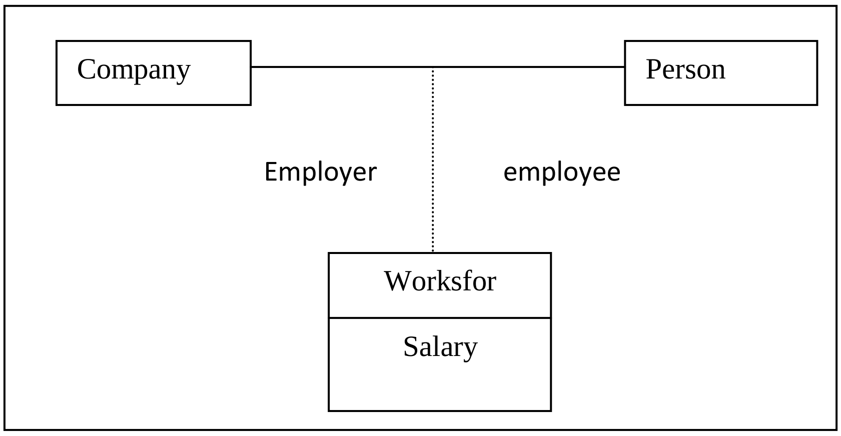
It is given for roles within associations. Sequence of integer intervals, where an interval represents a range of integers in this format. 0…1(lower bound …..upper bound) 0…* 1..3, 7…10, 15, 19…*



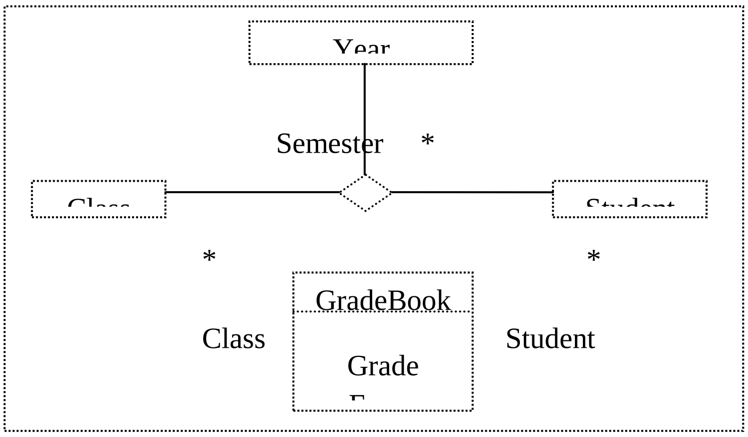
OR association – dashed line connecting two or more associations.



Association class – association that has class properties.

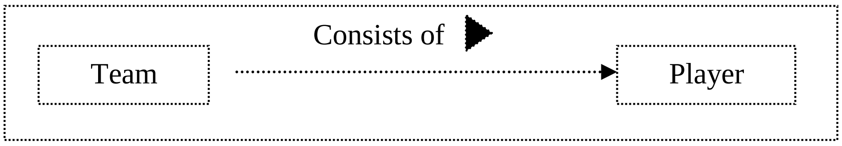


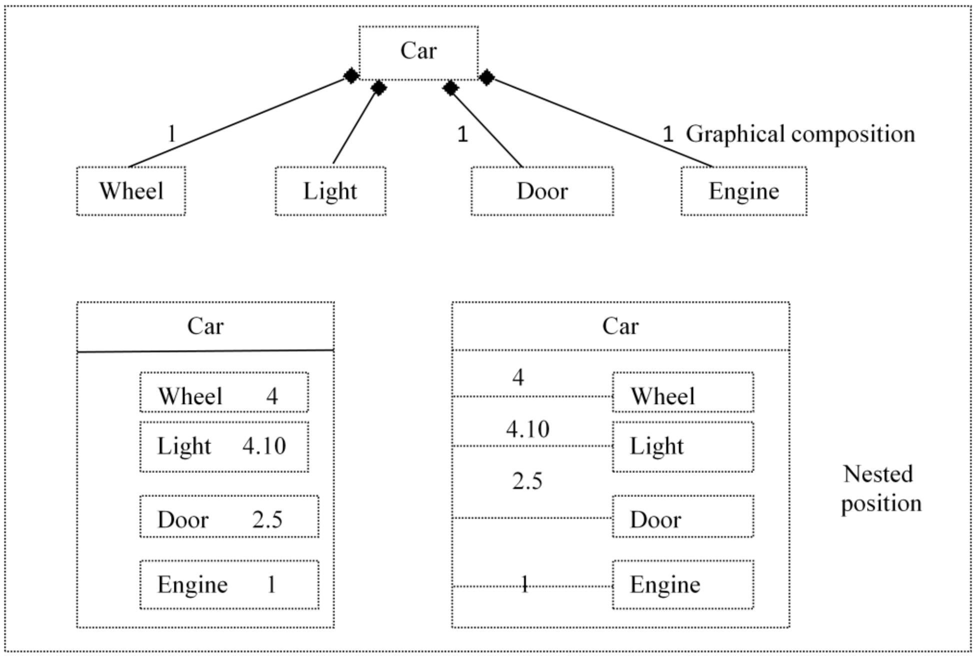
NOTES



N-ary association – association among more than two classes.

Aggregation and composition– hollow diamond attached to the end of the path. Aggregation is a form of association. Composition also known as the *a-part-of* is a form of aggregation with strong ownership to represent the component of a complex object.





Generalization – relationship between a general class and a more specific class.

6.8 UML DYNAMIC MODELING

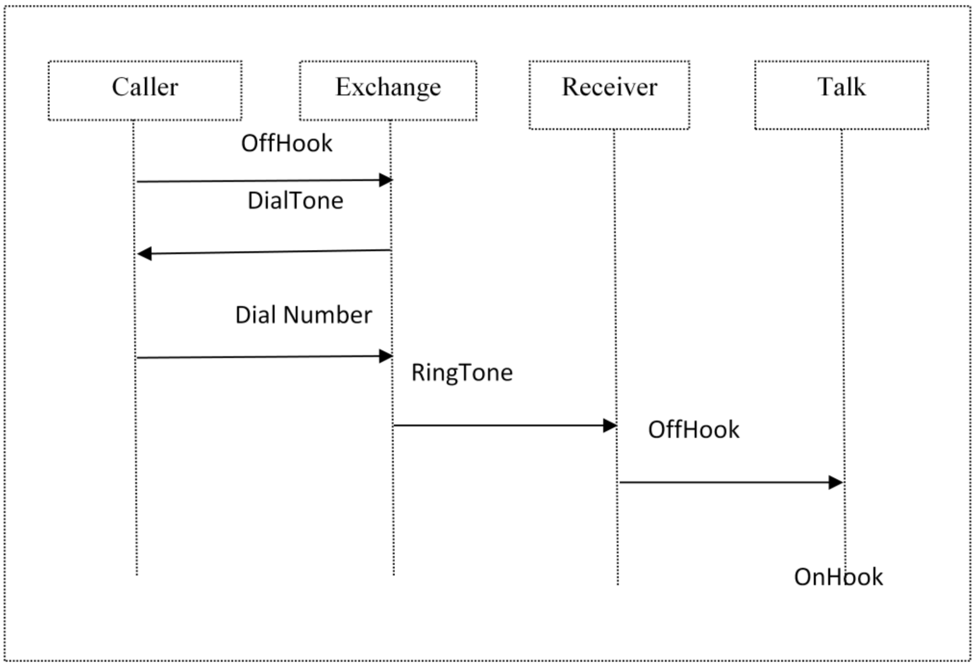
UML Interaction Diagrams

Interactions diagram describes how group of objects collaborate to get the job done.

It captures the behavior of the single use case showing the pattern of interaction among objects.

UML Sequence Diagrams

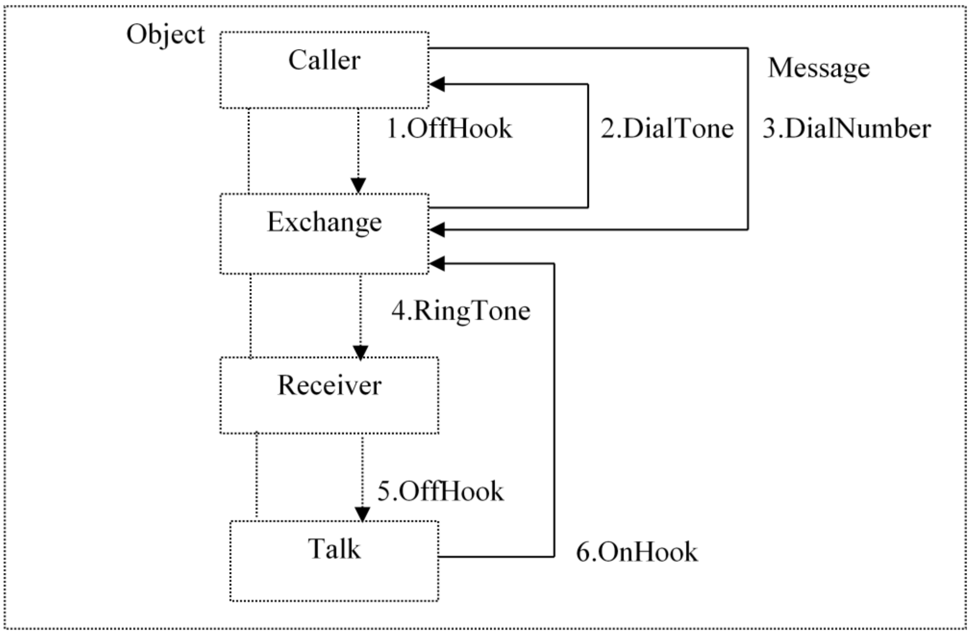
- Sequence diagrams illustrate how objects interact with each other.
- They focus on message sequences, that is, how messages are sent and received between a number of objects.
- Sequence diagrams have two axes: the vertical axis shows time and the horizontal axis shows a set of objects.
- The Instance form describes a specific scenario in detail
- The Generic form describes all possible alternatives in a scenario, therefore branches, conditions, and loops.



UML Collaboration Diagram

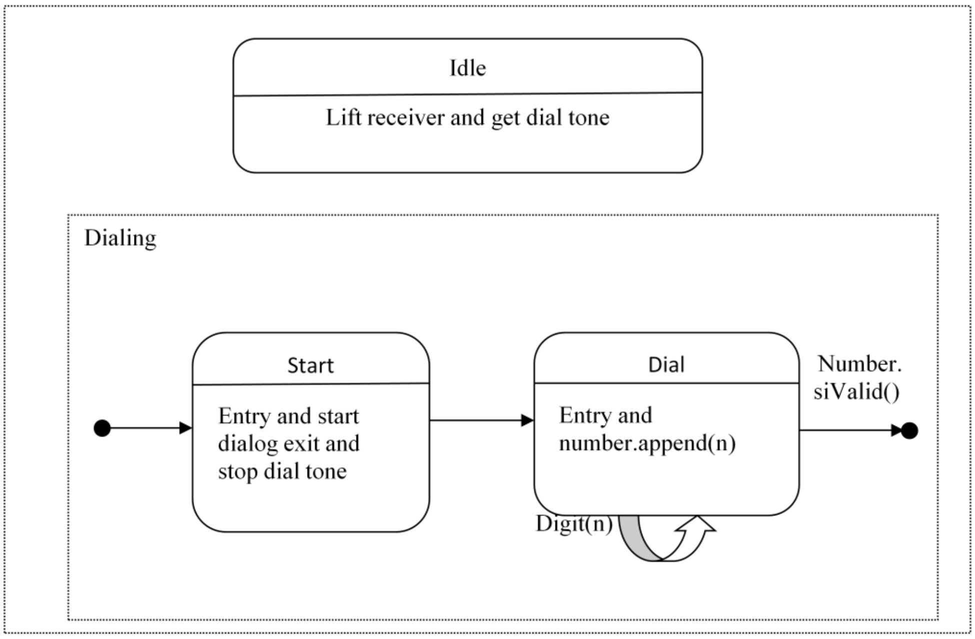
Collaboration diagrams focus on the interaction and the line between a set of collaborating objects. The sequence diagram focuses on time but the collaboration diagram focuses on space.

NOTES



UML State Chart Diagram

A State chart diagram shows the sequence of states that an object goes through during its life in response to outside and messages. The state is a set of values that describes an object at a specific point in time and is represented by state symbols and the transitions are represented by rows connecting the state symbols. A state chart diagram may contain sub diagrams. A state diagram represents the state of the method execution and activities in the diagram represent the activities of the object that performs the method. The purpose of the state diagram is to understand the algorithm involved in performing the method. To complete the OOD, the activities within the diagram must be assigned to objects and the control flows assigned to links in the object diagram.

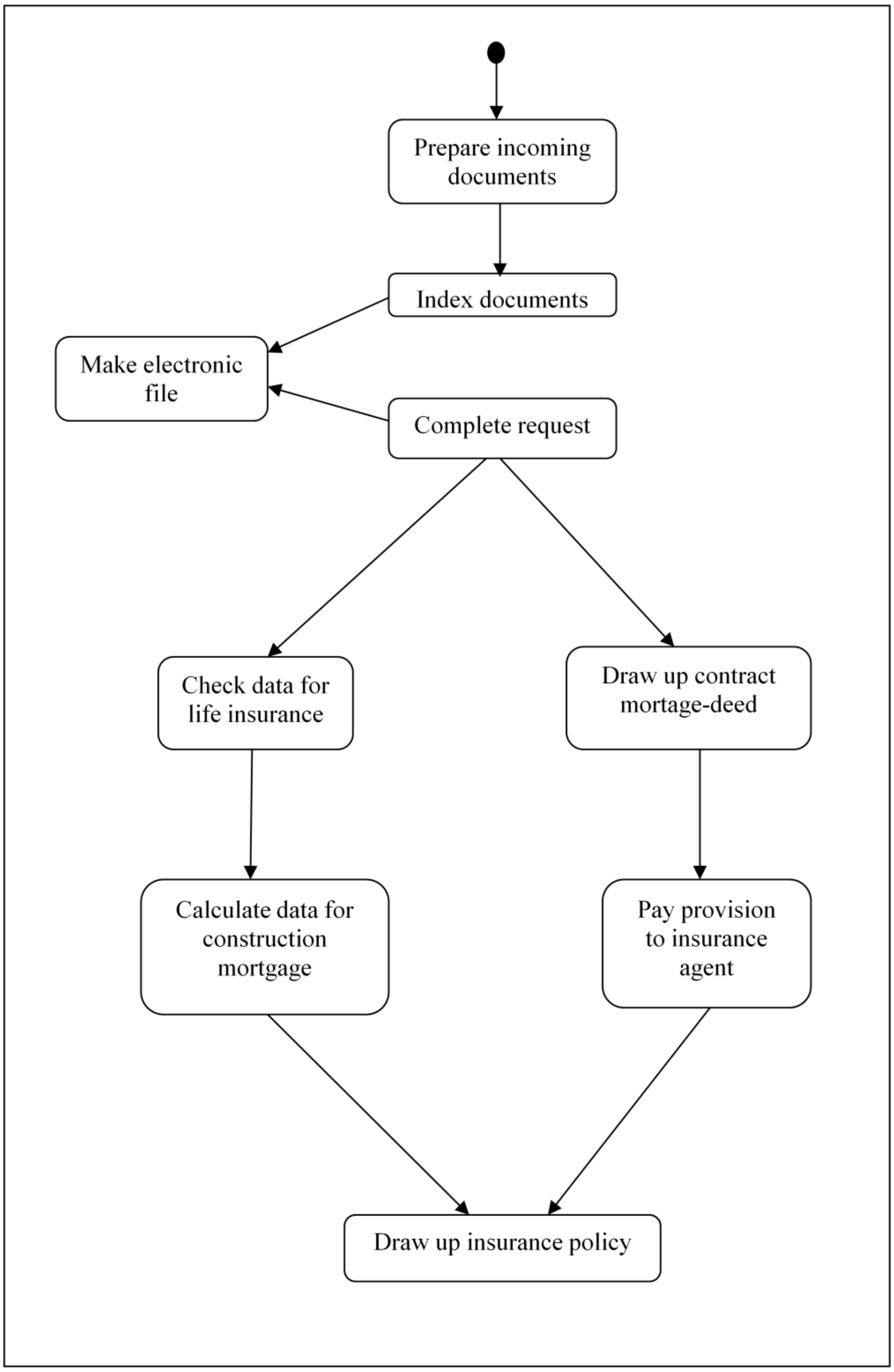

Idle
Lift receiver and get dial tone
Dialing
Start
Entry and start dialog exit and stop dial tone
Dial
Entry and number.append(n)
Number. siValid()
Digit(n)

UML ACTIVITY DIAGRAM

An activity diagram is a variation or special case of a state machine, in which the states are activities representing the performance of operations

NOTES

and the transitions are triggered by the completion of the operations. An activity diagram is used mostly to show the internal state of an object, but external events may appear in them. An external event appears when the object is in a "wait state" a state during which there is no internal activity by the object and the object is waiting of some external event to occur as a result of an activity of an another object. Activity and state diagrams express a decision when conditions are used to indicate different possible transitions that depend on Boolean conditions of container object.


Prepare incoming documents
Index documents
Make electronic file
Complete request
Check data for life insurance
Draw up contract mortage-deed
Calculate data for construction mortgage
Pay provision to insurance agent
Draw up insurance policy

NOTES

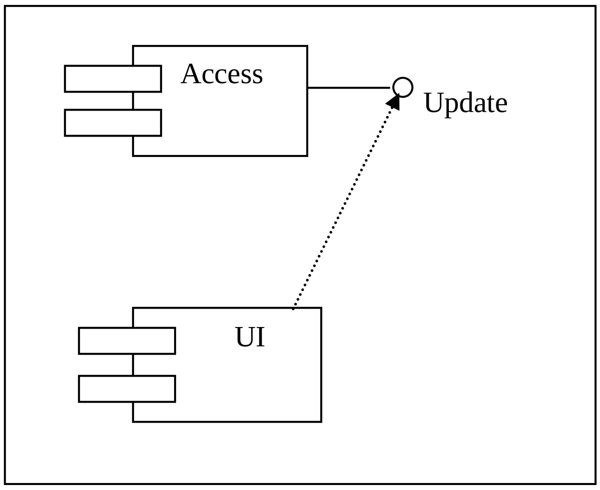
Implementation Diagrams

These diagrams show the implementation phase of systems development, such as the source code structure and the run-time implementation structure.

There are two types of implementation diagrams:

- Component diagrams show the structure of the code itself.
- Deployment diagrams show the structure of the run-time system.

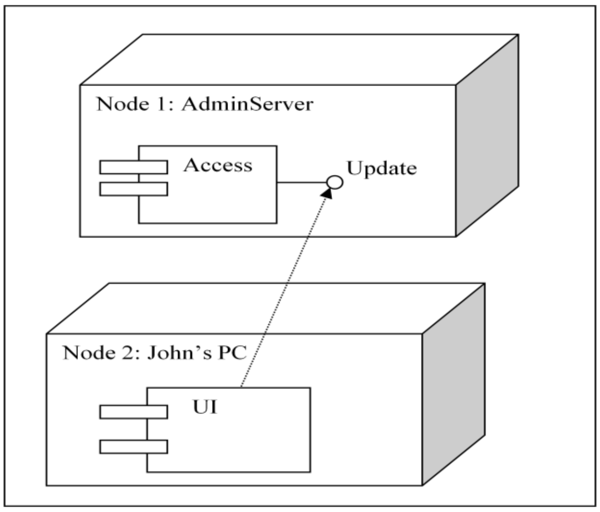
Component diagram



Component diagrams model the physical components such as (source code, exe, UI) in a design. Components are connected by dependency relationships. Component is represented by the boxed figure shown in above fig, Dependency is shown as a dashed arrow.

Deployment diagram

This diagram shows the configuration of run-time processing elements and the software components, process, and objects live in the system. Deployment diagrams show how physical modules of code are distributed on various hardware platforms.



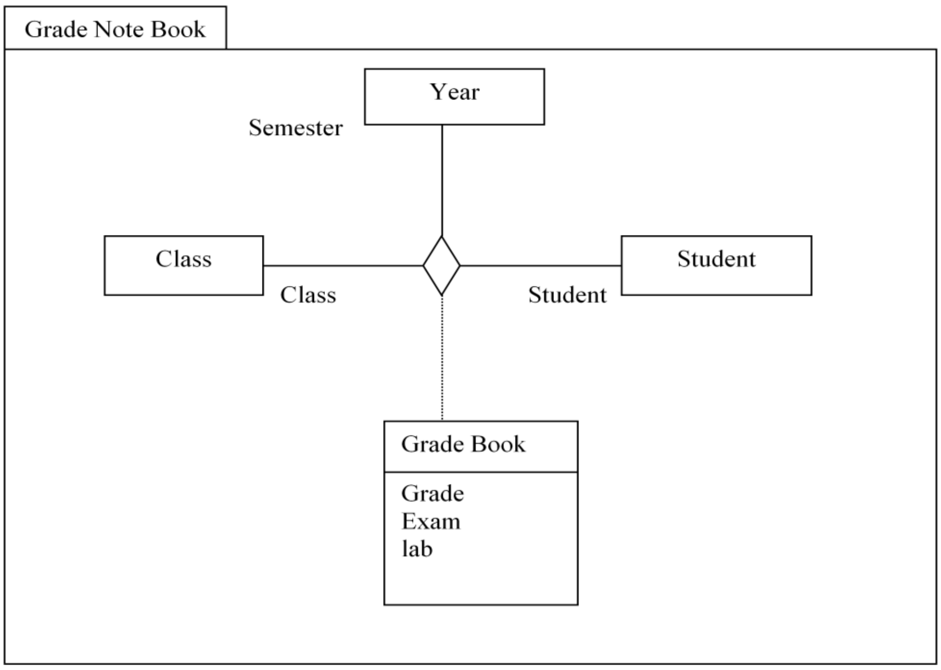
Check Your Progress

4. What is class interface notation?
5. Define Interaction diagram.
6. Give the purpose of state diagram
7. Define Activity diagram.
8. List the types of implementation diagram.

6.9 PACKAGE

A package is a grouping of model elements.

- Packages themselves may contain other packages.
- A package may contain both subordinate packages and ordinary model elements
- A package is a represented as folder, shown as a large rectangle with a tab attached to its upper left corner
- If contents of the package are not shown, than the name of the package is placed within the large rectangle.
- If contents of the package are shown, then the name of the package may be placed on the tab.



6.10 UML EXTENSIBILITY

Model Constraints and comments

Constraints are assumptions or relationships among model elements specifying conditions and propositions that must be maintained as true otherwise the system described by the model would be invalid.

Note

A note is a graphic symbol containing textual information; it also could contain embedded images.

Stereotypes

Stereotypes represent a built-in extendibility mechanism of the UML.

- User-defined extensions of the UML are enabled through the use of stereotypes and constraints.

6.11 UML META-MODEL

A meta-model is a model of modelling elements. The purpose of the UML meta-model is to provide a single, common, and definitive statement of the syntax and semantics of the elements of the UML. The UML meta-model describes the relationship between association and generalization. Association is depicted as composition of association roles. Here, we use UML modelling elements (such as generalization and composition) to describe the model itself, hence, the term meta-model.

Check Your Progress

9. How a package is represented in diagrams.
10. Define Note.
11. What is the purpose of meta model?

6.12 ANSWERS TO CHECK YOUR PROGRESS QUESTIONS

1. The purpose of access layer is to translate the requests and translate the results.
2. Static model can be viewed as a snapshot of a system’s parameters at rest or at a specific point in time Ex: Class diagram
3. Dynamic model can be viewed as a collection of procedures or behaviours that taken together reflect the behaviour of a system over time ex: Interaction diagram.
4. Class interface notation is a small circle with the interface name connected to the class.
5. Interactions diagram describes how group of objects collaborate to get the job done.
6. The purpose of the state diagram is to understand the algorithm involved in performing the method.
7. An activity diagram is a variation or special case of a state machine, in which the states are activities representing the performance of operations and the transitions are triggered by the completion of the operations.

8. The two types of implementation diagrams are Component diagram and Deployment diagram.
9. A package is a represented as folder, shown as a large rectangle with a tab attached to its upper left corner
10. A note is a graphic symbol containing textual information; it also could contain embedded images.
11. The purpose of the UML meta-model is to provide a single, common, and definitive statement of the syntax and semantics of the elements of the UML.

6.13 SUMMARY

- The main motivation for going to unified approach is to combine the best practices, processes, methodologies, and guidelines along with UML (Unified Modeling Language) notations and diagrams for better understanding object oriented concepts and system development.
- A model is an abstract representation of a system, constructed to understand the system prior to building or modifying it.
- Static model can be viewed as a snapshot of a system's parameters at rest or a specific point in time.
- It can be viewed as a collection of procedures or behaviours that taken together reflect the behaviour of a system over time.
- Modeling reduces complexity.
- The unified modeling language (UML) is a language for specifying, constructing, visualizing and documenting the software system and its components.
- A package groups and manages the modeling elements, such as classes, their associations, and their structures.
- Implementation diagrams show the implementation phase of system development, such as the source code structure and the run-time implementation structure.

6.14 KEYWORDS

- Object oriented Analysis
- Object oriented Testing
- Repository
- Modelling
- Class Diagram
- UML
- Package
- Meta model

NOTES

6.15 REVIEW QUESTIONS

1. Define model. Explain about the types of model.
2. What are the advantages of Modeling?
3. Define UML.
4. Mention the primary goals in the design of the UML.
5. Give the nine UML graphical diagrams.
6. Write short note on UA proposed Repository.
7. How interfaces are represented in UML?
8. Explain in detail about the Unified approach?
9. Describe the UML Class diagram?
10. Explain the Interaction diagrams with an example.

6.16 FURTHER READINGS

1. Booch G.,"Object Oriented Analysis And Design",Addison-Wesley Publishing Company,1994.
2. Rambaugh J, Blaha.M. Premeriani, W., Eddy F and Loresen W., "Object Oriented Modeling and Design", PHI, 1997.

NOTES

BLOCK 3: OBJECT ORIENTED ANALYSIS
UNIT VII: OBJECT ORIENTED ANALYSIS

Structure

7.0 INTRODUCTION

The first step in finding an appropriate solution to a given problem is to understand the problem and its domain. Analysis is the process of transforming a problem definition from a fuzzy set of facts and myths into a coherent statement of a system's requirements. Analysis involves a great deal of interaction with the people who will be affected by the system, including the actual users and anyone else on which its creation will have an impact. An object oriented environment allows the same set of models used for analysis, design and implementation.

7.1 OBJECTIVES

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

- Understand use cases
- Describe use case models
- Understand about documentation

7.2 IDENTIFYING USE CASES

The use-case approach to object-oriented analysis and the object-oriented analysis process consists of

- Identifying actors.
- Identifying use cases.
- Documentation.

What Is Analysis?

Analysis is the process of transforming a problem definition from a fuzzy set of facts and myths into a coherent statement of a system‘s requirements. The main objective of the analysis is to capture a complete, unambiguous, and consistent picture of the requirements of the system and what the system must do to satisfy the users' requirements and needs.

Requirements Difficulties

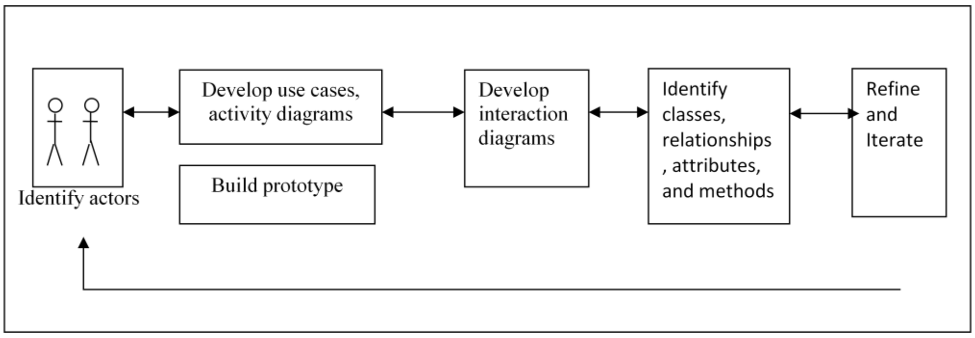
Three most common sources of requirements difficulties are:

1. Incomplete requirements.
2. Fuzzy descriptions (such as fast response).
3. Unneeded features.

The Object-Oriented Analysis (OOA) Process

The process consists of the following steps:

1. Identify the actors: i.Who is using the system? ii.Or, in the case of a new system, who will be using the system?
2. Develop a simple business process model using UML activity diagram.
3. Develop the use case
   a. What the users are doing with the system?
   b. Or, in the case of a new system, what users will be doing with the system? Use cases provide us with comprehensive documentation of the system under study.
4. Prepare interaction diagrams:
   - Determine the sequence.
   - Develop collaboration diagrams
5. Classification — develop a static UML class diagram
   - Identify classes.
   - Identify relationships.
   - Identify attributes.
   - Identify methods.
6. Iterate and refine: If needed, repeat the preceding steps.


Identify actors
Develop use cases, activity diagrams
Build prototype
Develop interaction diagrams
Identify classes, relationships, attributes, and methods
Refine and Iterate

7.3 DEVELOPING BUSINESS PROCESSES MODELLING

Developing an activity diagram of the business processes can provide us with an overall view of the system. It is not necessary for all projects. When required business process and requirements can be modelled to any level of detail. Activity diagram support this modelling.

Disadvantages - Time consuming process

Advantages – Familiarity

Check Your Progress

1. Give the objective of analysis.
2. What are the steps in preparing interaction diagrams?

7.4 USE CASE MODEL

Use cases are scenarios for understanding system requirements. The use-case model describes the uses of the system and shows the courses of events that can be performed.

Use case defines what happens in the system when a use case is performed. The use case model tries to systematically identify uses of the system and therefore the system's responsibilities.

7.4.1 Use Cases Under the Microscope:

A Use Case is a sequence of transactions in a system whose task is to yield results of measurable value to an individual actor of the system.

Use Case Key Concepts

- Use case - Use case is a special flow of events through the system.
- Actors - An actor is a user playing a role with respect to the system.
- In a system - This simply means that the actors communicate with the system's use case.
- A measurable value - . A use case must help the actor to perform a task that has some identifiable value.

- Transaction - A transaction is an atomic set of activities that are performed either fully or not at all.

7.4.2 Use Associations

The use association occurs when you are describing your use cases and notice that some of them have common sub flows. The use association allows you to extract the common sub flow and make it a use case of its own.

Extends Associations

The extends association is used when you have one use case that is similar to another use case but does a bit more or is more specialized; in essence, it is like a subclass.

Types of Use Cases

- Use cases could be viewed as concrete or abstract.
- An abstract use case is not complete and has no initiation actors but is used by a concrete use case, which does interact with actors.

7.4.3 Identifying the Actors

i. The term actor represents the role a user plays with respect to the system.

ii. When dealing with actors, it is important to think about roles rather than people or job titles.

iii. Who affects the system? Or,

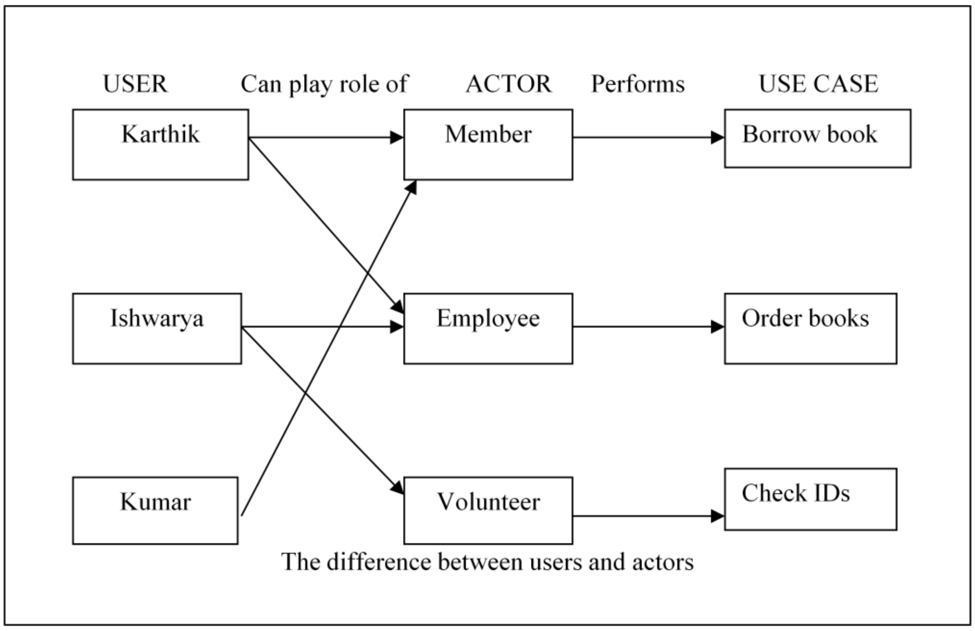
iv. Which user groups are needed by the system to perform its functions? These functions can be both main functions and secondary functions, such as administration.

v. Which external hardware or other systems (if any) use the system to perform tasks?

vi. What problems does this application solve (that is, for whom)?

vii. And, finally, how do users use the system (use case)? What are they doing with the system?

NOTES



The difference between users and actors

7.4.4 Guidelines for Finding Use Cases

- For each actor, find the tasks and functions that the actor should be able to perform or that the system needs the actor to perform.
- Name the use cases.
- Describe the use cases briefly by applying terms with which the user is familiar.

Separate Actors from Users

- Each use case should have only one main actor
- Isolate users from actors.
- Isolate actors from other actors (separate the responsibilities of each actor).
- Isolate use cases that have different initiating actors and slightly different behaviour.

7.4.5 How detailed must a use case be? When to stop decomposing it and when to continue

- Develop system use case diagram
- Draw package
  – to represent business domains of the system for each package
- create child use case diagram
- Prepare at least one scenario for each use case
  - Each scenario shows different sequence of interaction , with all decisions definite
- When the lowest use case level is arrived, which can't be broken further,

NOTES

- sequence and collaboration diagram is drawn

7.4.6 Dividing use case into package

- Whole system is divided into many packages
- Each package encompasses multiple use cases
- Each use-case represent a scenario in the system
- a design can be broken down into packages, and each of packages consists multiple use-cases

Check Your Progress

3. Define use case model.
4. Define actor.
5. Define transaction.
6. List the types of use cases.

7.5 DOCUMENTATION

An effective document can serve as a communication vehicle among the project's team members, or it can serve as initial understanding of the requirements.

- Important factor in making a decision about committing (assigning, handover, giving) resources
- Mainly depends on rules and regulation

7.5.1 Guidelines for Effective Documentation:

1. Common Cover - All documents should share a common cover sheet that identifies the document, the current version, and the individual responsible for the content
2. 80-20 Rule - 80 percent of the work can be done with 20 percent of the documentation.
3. The trick is to make sure that the 20 percent is easily accessible and the rest (80percent) is available to those (few) who need to know.

Familiar Vocabulary

- Use a vocabulary that your readers understand and are comfortable with.
- The main objective here is to communicate with readers and not impress them with buzz words.

Make the Document as Short as Possible

- Eliminate all repetition;
- Present summaries, reviews, organization chapters in less than three pages.

- Make chapter headings task oriented so that the table of contents also could serve as an index.

Organize the Document

Use the rules of good organization within each section.

The main objective of the analysis is to capture a complete, unambiguous, and consistent picture of the requirements of the system.

To construct several models and views of the system to describe what the system does rather than how.

Capturing use cases is one of the first things to do in coming up with requirements. Every use case is a potential requirement.

The key in developing effective documentation is to eliminate all repetition; present summaries, reviews, organization chapters in less than three pages.

Check Your Progress

7. What is the advantage of effective documentation?
8. What is the key in developing effective documentation?

7.6 ANSWERS TO CHECK YOUR PROGRESS QUESTIONS

1. The main objective of the analysis is to capture a complete, unambiguous, and consistent picture of the requirements of the system and what the system must do to satisfy the users' requirements and needs.
2. First determine the sequence and then to develop collaboration diagrams
3. The use-case model describes the uses of the system and shows the courses of events that can be performed.
4. An actor is a user playing a role with respect to the system.
5. A transaction is an atomic set of activities that are performed either fully or not at all.
6. Use cases could be viewed as concrete or abstract.
7. An effective document can serve as a communication vehicle among the project's team members, or it can serve as initial understanding of the requirements.
8. The key in developing effective documentation is to eliminate all repetition; present summaries, reviews, organization chapters in less than three pages.

NOTES

7.7 SUMMARY

- Analysis is the process of transforming a problem definition from a fuzzy set of facts and myths into a coherent statement of a system‘s requirements.
- A use-case model is a model of the system’s intended functions (use cases) and its surroundings (actors).
- Use cases are scenarios that describe how actors use the system.
- An actor represents anything that interacts with the system.
- Use cases could be viewed as concrete or abstract.
- An effective document can serve as a communication vehicle among the project's team members, or it can serve as initial understanding of the requirements.

7.8 KEYWORDS

- Use case model
- Associations
- Actors
- Documentation

7.9 REVIEW QUESTIONS

1. What is a use case model?
2. Describe the basic activities in OO analysis?
3. Who are actors?
4. Why are uses and extends associations useful in use case modeling?
5. How do you identify actors?
6. What is the difference between users and actors? How would you identify them?
7. What is 80-20 rule?
8. What are the guidelines for developing effective documentation?

7.10 FURTHER READINGS

1. Bahrami, A.(1999). Object Oriented Systems Development, using the unified modeling language, McGraw-Hill
2. Object Oriented Analysis and Design using UML, by Rational Software Corporation
3. Dennis,A., Wixon,B.H., and Tegarden,D.(2002). Systems Analysis and Design : An Object Oriented Approach with UML

UNIT VIII: CLASSIFICATION

Structure

8.0 INTRODUCTION

- Identification of classes is the hardest part of part of OO-Analysis.
- Object analysis is a process by which we can identify the classes that play a role in achieving the system goals & requirements
- Booch states that " There is no such a thing as the perfect class structure, nor the right set of objects"
- Classification is the categorization of input data (things) into identifiable classes through the extraction of significant features of attributes of the data from a background of irrelevant detail
- Classes are an important mechanism for classifying objects
- The main role of a class is to define the attributes, methods, and applicability of its instances.

8.1 OBJECTIVES

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

- Describe the approaches of identifying classes
- Understand Noun phrase approach
- Understand Common class pattern approach

NOTES

8.2 APPROACHES FOR IDENTIFYING CLASSES

There are four approaches to identify classes. They are

- Noun phrase approach
- Common class patterns approach
- Use-case driven approach
- Classes, responsibilities, & collaborators (CRC) approach

8.3 NOUN PHRASE APPROACH

- Proposed by Rebecca Wirfs-Brock, Brian Wilkerson, & Lauren Wiener
- Read through the requirements or use-cases looking for noun phrases
- Nouns in textual description are considered as classes
- Verbs are considered as methods
- All plurals are changed to singular

Nouns are listed and divided into three categories

- Relevant classes
- Fuzzy classes
- Irrelevant classes

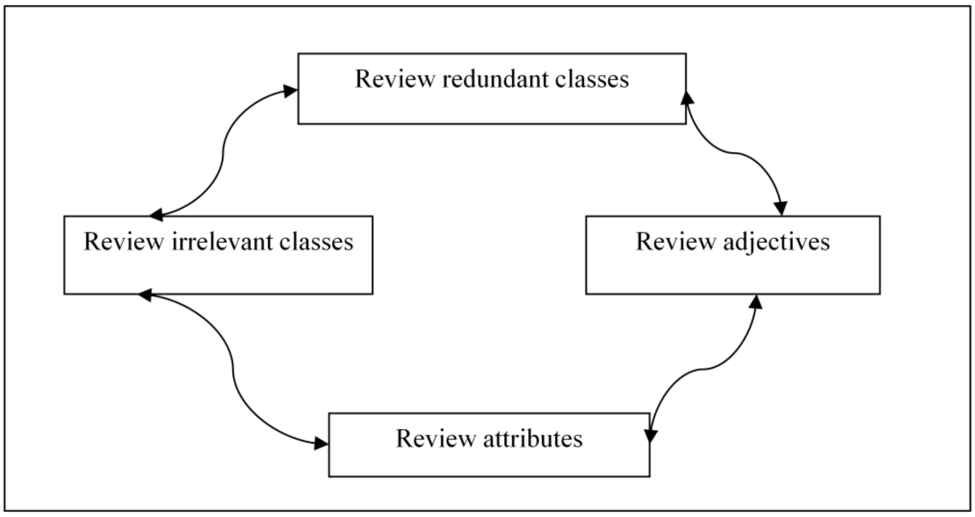
8.3.1 Guidelines for selecting classes in an application

- Look for nouns and noun phrases in the use-cases
- Some classes are implicit or taken from general knowledge
- All classes must make sense in the application domain; avoid computer implementation classes – defer them to the design stage
- Carefully choose and define class names

8.3.2 Guidelines in selecting candidate classes from the relevant and fuzzy categories

- Redundant classes do not keep two classes that express the same information
- Adjectives classes can suggest a different kind of object, different use of the same object, or it could be utterly irrelevant. If the use of the adjective signals that the behaviour of the object is different, then make a new class
- Attribute classes – Tentative objects that are used only as values should be defined or restates as attributes and not as a class

- Irrelevant classes - Each class must have a purpose and should be clearly defined and necessary. Formulate a statement of purpose for each candidate class



The process of eliminating the redundant classes and refining the remaining classes is not sequential. Can move back and forth among these steps

8.3.3 Initial list of noun classes : in vianet bank

- Account
- Account balance
- Amount
- Approval process
- Atm card
- Atm machine
- Bank
- Bank client
- Card
- Cash
- Check
- Checking
- Checking account
- Client
- Client’s account
- Currency
- Dollar
- Envelope
- Four digits
- Fund
- Invalid pin
- Message
- Money
- Password
- PIN
- Pin code

- Record
- Savings
- Savings account
- Step
- System
- Transaction
- Transaction history

8.3.4 Removing irrelevant classes

- Account
- Account balance
- Amount
- Approval process
- Atm card
- Atm machine
- Bank
- Bank client
- Card
- Cash
- Check
- Checking
- Checking account
- Client
- Client's account
- Currency
- Dollar
- Envelope
- Four digits
- Fund
- Invalid pin
- Message
- Money
- Password
- PIN
- Pin code
- Record
- Savings
- Savings account
- Step
- System
- Transaction
- Transaction history

8.3.5 Reviewing the Redundant classes and Building a common vocabulary

- Account
- Account balance
- Amount

- Approval process
- Atm card
- Atm machine
- Bank
- Bank client
- Card
- Cash
- Check
- Checking
- Checking account
- Client
- Client’s account
- Currency
- Dollar
- Envelope
- Four digits
- Fund
- Invalid pin
- Message
- Money
- Password
- PIN
- Pin code
- Record
- Savings
- Savings account
- Step
- System
- Transaction
- Transaction history

8.3.6 Reviewing the class purpose

- Include classes with
- Purpose
- Clear definition
- Necessary in achieving system goal
- Eliminate classes with no purpose
- Ex: Candidate class with purpose are
- ATM machine class
- ATM card class
- Bankclient class
- Bank class
- Account class
- Checking account class
- Saving account class
- Transaction class

8.4 COMMON CLASS PATTERNS APPROACH

It was proposed by Shaler &Mellor, Ross, and Coad & Yourdon

They have listed the following patterns for finding the candidate class and object

Concept class: A concept is a particular idea or understanding that we have of our world. It consists of principles that are not tangible but used to organize or keep track business activities or communications.

Ex: performance

Events class: They are points in time that must be recorded. Associated with things remembered are attributes such as who, what, when, where, how, or why.

Ex : landing, order, request

Organization class: A collection of people, resources, facilities, or groups to which users belong, and their capabilities have a defined mission, whose existence is largely independent of individuals.

Ex : human resources department

People class: Known as person, roles, and roles played class. Represents the different roles users play in interacting with the application. What role does a person play in the system ?

Ex : employee, student, lecturer

Places class: Physical locations that the system must keep information about

Ex : buildings, stores, offices

Tangible things and devices class: Includes physical objects or groups of objects that are tangible and devices with which the application interacts

Ex : car (tangible), pressure sensors (devices).

Check Your Progress

1. What are the approaches available to identify classes?
2. What is a redundant class?
3. Define attribute classes.
4. What is an organization class?

8.5 ANSWERS TO CHECK YOUR PROGRESS QUESTIONS

1. There are four approaches to identify classes. They are Noun phrase approach, Common class patterns approach, Use-case driven approach, Classes, responsibilities, & collaborators (CRC) approach.

2. Redundant classes do not keep two classes that express the same information
3. Attribute classes are tentative objects that are used only as values should be defined or restates as attributes and not as a class.
4. An organization class is a collection of people, resources, facilities, or groups to which users belong, and their capabilities have a defined mission, whose existence is largely independent of individuals.

8.6 SUMMARY

- Identification of classes is the hardest part of part of OO Analysis.
- Classification is the categorization of input data (things) into identifiable classes through the extraction of significant features of attributes of the data from a background of irrelevant detail.
- The class name should be singular
- One general rule for naming classes is that use names with which the users or clients are comfortable
- The name of a class should reflect its intrinsic nature.,
- Use readable name, Capitalize class names.
- There are four approaches to identify classes.
- Nouns in textual description are considered as classes
- Nouns are listed and divided into three categories, relevant classes, fuzzy classes, irrelevant classes
- Concept class concept is a particular idea or understanding that we have of our world
- Events class is points in time that must be recorded
- Organization class is a collection of people, resources, facilities, or groups to which users belong, and their capabilities have a defined mission, whose existence is largely independent of individuals.
- People class is known as person, roles, and roles played class
- Places class is a physical location that the system must keep information about
- Tangible things and devices class which includes physical objects or groups of objects that are tangible and devices with which the application interacts.

8.7 KEYWORDS

- Classification
- Nouns
- Organization class

- Common class
- Redundant class

8.8 REVIEW QUESTIONS

1. What are the approaches to identify classes?
2. Describe the noun phrase strategy for identifying tentative classes in the problem domain?
3. Describe relevant, fuzzy, irrelevant classes.
4. How do you select candidate classes from the list of relevant and fuzzy classes?
5. How to identify classes? Explain.
6. What are event classes?
7. What are organization classes?
8. What are people classes?
9. Write a note on common class patterns approach.
10. Explain noun phrase approach in detail.

8.9 FURTHER READINGS

1. Booch G.,"Object Oriented Analysis And Design",Addison-Wesley Publishing Company,1994.
2. Anderson, Michael and Bergstrand, John, "Formalizing Use Cases with Message Sequence Charts", 1995.

UNIT IX: Use Case Driven Approach

Structure

9.0 INTRODUCTION

All objects stand in relationship to others on whom they rely for services and control. The relationship among objects is based on the assumptions each makes about the other objects, including what operations can be performed and what behaviour results.

The three types of relationships among objects are

Association

Super-Sub structure (also known as generalization hierarchy)

Aggregation and a-part-of structure.

9.1 OBJECTIVES

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

- Understand about associations
- Describe relationships
- Describe responsibility
- Understand class diagrams

9.2 ASSOCIATION

Association represents a physical or conceptual connection between two or more objects. Binary associations are shown as lines connecting two class symbols. Ternary and higher-order associations are shown as diamonds connecting to a class symbol by lines, and the association name is written above or below the line.

9.2.1 Identifying Associations

We need to identify the system's responsibilities because responsibilities identify problems that are to be solved. A responsibility serves as a handle for discussing potential solutions. Once the system's responsibilities are understood, we can start identifying the attributes of the system's classes.

The following questions can assists in identifying associations [Wirfs-Brock, ilkerson, &Wiener]:

- Is the class capable of fulfilling the required task by itself?
- If Not, what does it need?
- From what other class can it acquire what it needs?

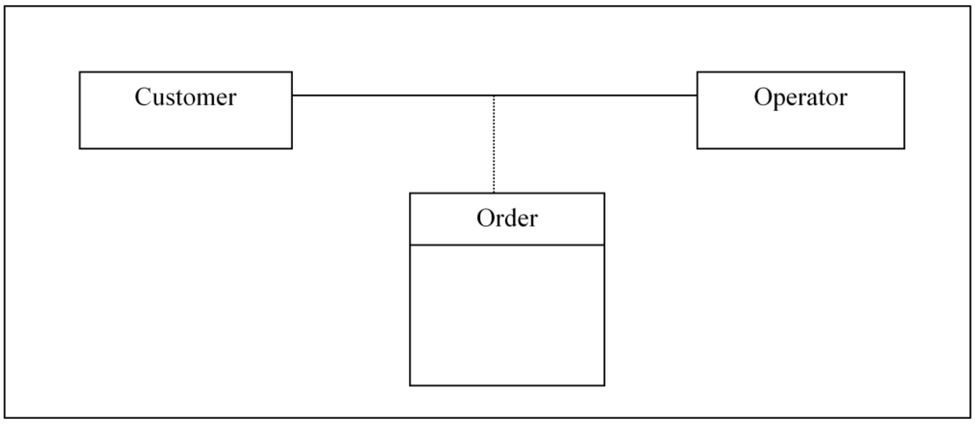
9.2.2 Guidelines for Identifying Association

- A dependency between two or more classes may be an association.
- A reference from one class to another is an association.

9.2.3 Common Association Patterns

Location Association – next to, part of, contained in.

Communication Association – talk to, order to.



9.2.4 Eliminate Unnecessary Associations

Implementation association: It is concerned with the implementation design of the class within certain programming or development environment and not relationships among business objects.

Ternary associations: It complicates the representation.

Directed actions associations: It can be defined in terms of other associations. Since they are redundant, it is avoided.

9.3 SUPER – SUB CLASS RELATIONSHIPS

A class is part of hierarchy of classes, where the top class is the most general one and from it descend all other, more specialized classes.

Represents the inheritance relationships between related classes

Also known as generalization hierarchy

Advantage

Can build on what already have and more important, reuse what already have

Inheritance allows classes to share and reuse behaviours and attributes

Example: Generalization hierarchy

9.3.1 Guidelines for Identifying Super-sub Relationships:

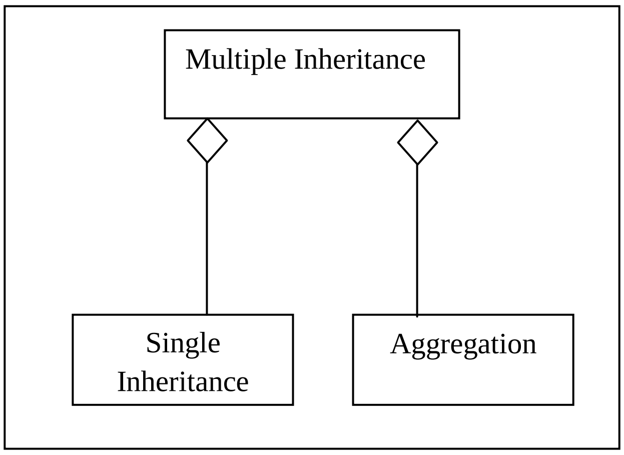
Top-down: Look for noun phrases composed of various adjectives on class name. Only specialize when the sub classes have significant behavior.

Example, Military Aircraft and Civilian Aircraft.

Bottom-up: Look for classes with similar attributes or methods. Group them by moving the common attributes and methods to super class. Do not force classes to fit a preconceived generalization structure.

Reusability: Move attributes and methods as high as possible in the hierarchy. At the same time do not create very specialized classes at the top of hierarchy. This balancing act can be achieved through several iterations.

Multiple inheritance: Avoid excessive use of multiple inheritances. It is also more difficult to understand programs written in multiple inheritance system.



Check Your Progress

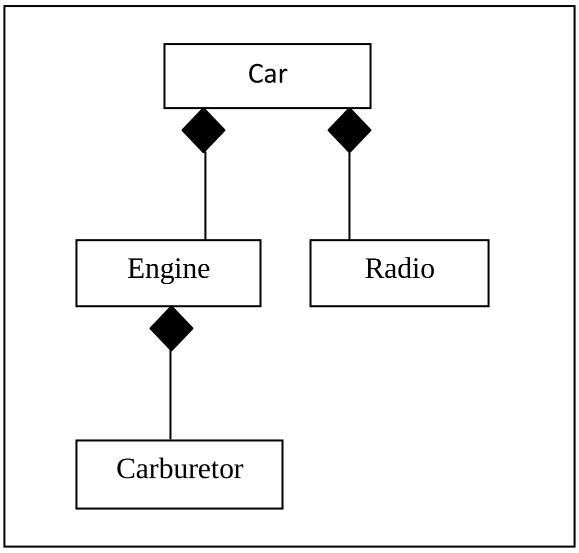
1. What is the use of responsibility?
2. Define implementation association.
3. How to identify relationship using top down approach?

9.4 A – PART – OF RELATIONSHIPS - AGGREGATION

Represents the situation where a class consists of several component classes

NOTES

A class that is composed of other classes does not behave like its part; actually ,it behaves very differently.



Two major properties of an aggregation are:

- Transitivity: the property where, if A is part of B and part B is part of C, then A is part of C. Ex : a carburetor is part of an engine, and engine is a part of a car, Therefore a carburetor is aprt of a car.
- Antisymmetry: The property where, if A is part of B, then B is not part of A.

Ex : An engine is aprt of car, BUT car is not part of an engine.

9.4.1 A – Part – of Relationships Patterns

Assembly: An assembly is constructed from its parts and an assembly part situation physically exists.

Container: A physical whole encompasses but it is not constructed from physical parts.

Collection member: A conceptual whole encompasses parts that may be physical or conceptual.

9.5 CLASS RESPONSIBILITY: IDENTIFYING ATTRIBUTES

- Attributes are things an object must remember
- An attribute is a data definition held by instances of a class (object)
- Attributes do not have behaviour, they are NOT objects
- Attribute are simple nouns or noun phrases
- names must be unique within a class
- Each attribute should have a clear, concise definition

- An attribute value is the value of an attribute for a particular object
- Each object has a value for every attribute defined for its class
- identifying attributes of a system's classes starts with understanding the systems responsibilities
- system's responsibilities can be identified by developing use cases and the desired characteristics of the applications, such as determining what information users need from the system
- The following questions help in identifying the responsibilities of classes and deciding what data elements to keep track of:
    - What information about an object should we keep track of to identify attributes of a class?
    - What services must a class provide to identify a class's methods?
- Many attributes are discovered in the flow of events for the use cases
- Look for nouns that were not considered good candidate classes
- Others are discovered when the class definition is created

9.6 CLASS RESPONSIBILITY: IDENTIFYING METHODS

Objects not only describe abstract data but also must provide some services

- In an object-oriented environment, every piece of data, or object is surrounded by a rich set of routines called methods. (These methods do everything from printing the object to initializing its variables)
- Operations (methods or behavior) in the object-oriented system usually correspond to queries about attributes (and sometimes association) of the objects
- An operation is a service that can be requested from an object to effect behavior
- Methods are responsible for managing the value of attributes such as query, updating, reading, and writing; Example : operation GetBalance, return the value of an account's balance
- Operation should be name to indicate their outcome NOT the steps behind the operation.

9.7 CLASS DIAGRAM

- A class diagram shows a set of classes, interfaces, and their relationships
- Model the static view of the system, which supports the functional requirements of the system

- Made up of the following basic elements :
  - Classes
  - Relationships
  - Associations
  - Aggregations
  - Generalizations

Check Your Progress

4. What is transitivity?
5. Define operations.

9.8 ANSWERS TO CHECK YOUR PROGRESS QUESTIONS

1. A responsibility serves as a handle for discussing potential solutions. Once the system's responsibilities are understood, we can start identifying the attributes of the system's classes.
2. Implementation association is concerned with the implementation design of the class within certain programming or development environment and not relationships among business objects.
3. Look for noun phrases composed of various adjectives on class name. Only specialize when the sub classes have significant behavior.
4. Transitivity is the property where, if A is part of B and part B is part of C, then A is part of C.
5. An operation is a service that can be requested from an object to effect behavior.

9.9 SUMMARY

- The three types of relationships among objects are Association, Super-Sub
- Structure, Aggregation and a-part-of structure.
- Association represents a physical or conceptual connection between two or more objects.
- Binary associations are shown as lines connecting two class symbols.
- Ternary and higher-order associations are shown as diamonds connecting to a class symbol by lines, and the association name is written above or below the line.
- A responsibility serves as a handle for discussing potential solutions.

- A-part-of relationship, also called aggregation, represents the situation where a class consists of several component classes.
- A class that is composed of other classes does not behave like its parts.
- Transitivity is the property where, if A is part of B and part B is part of C, then A is part of C.
- Antisymmetry is the property where, if A is part of B, then B is not part of A.

9.10 KEYWORDS

- Association
- Relationship
- Aggregation
- Attributes
- Methods
- Antisymmetry
- Transitivity

9.11 REVIEW QUESTIONS

1. What is association?
2. What is generalization?
3. How would you identify a super- subclass structure?
4. What is an a-part-of structure? Write their properties.
5. What guidelines would you use to identify a-part-of structure?
6. Is association different from an a-part-of relation?
7. What are unnecessary associations? How would you know?
8. How do you identify attributes?
9. How do you identify methods?
10. What are the unnecessary attributes?
11. Why do we need to justify classes with one attribute?
12. List the guidelines for identifying super-sub relationships?

9.12 FURTHER READINGS

1. Bahrami, A.(1999). Object Oriented Systems Development, using the unified modeling language, McGraw-Hill
2. Object Oriented Analysis and Design using UML, by Rational Software Corporation (2002)

BLOCK 4: UNIT X: OBJECT ORIENTED DESIGN
UNIT X: OBJECT ORIENTED DESIGN

Structure

10.0 INTRODUCTION

Main focus of the analysis phase of SW development is “what needs to be done”. Objects discovered during analysis serve as the framework for design. Class’s attributes, methods, and associations identified during analysis must be designed for implementation as a data type expressed in the implementation language.

During the design phase, we elevate the model into logical entities, some of which might relate more to the computer domain (such as user interface, or the access layer) than the real world or the physical domain (such as people or employees). Start thinking how to actually implement the problem in a program. The goal is to design the classes that we need to implement the system. Design is about producing a solution that meets the requirements that have been specified during analysis.

10.1 OBJECTIVES

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

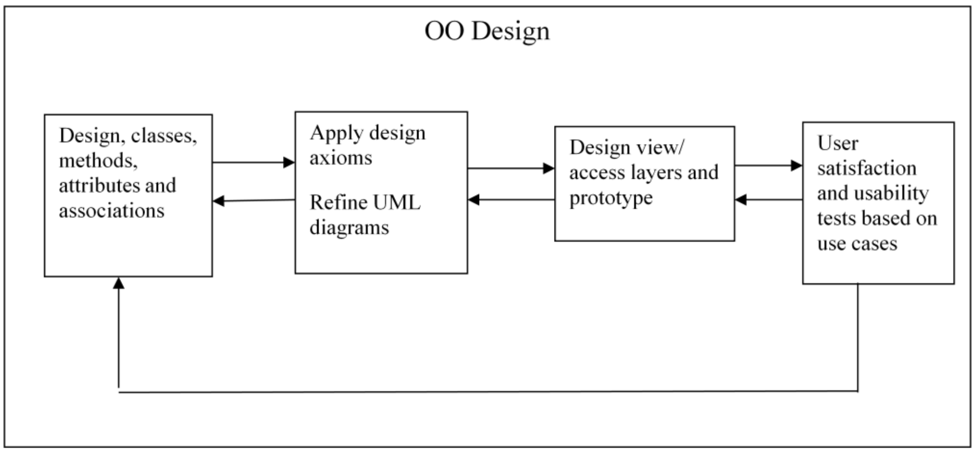
- Understand OOD design process and axioms
- Describe axioms of OOD
- Understand about corollaries
- Understand the design patterns and classes
- Understand UML attribute presentation
- Understand the design issues

10.2 OBJECT-ORIENTED DESIGN PROCESS AND DESIGN AXIOMS

•Analysis Phase
  –Class's attributes, methods and associations are identified
  –Physical entities, players and their cooperation are identified
  –Objects can be individuals, organizations or machines
•Design Phase
  –Using Implementation language appropriate data types are assigned
  –Elevate the model into logical entities (user interfaces)
  –Focus is on the view and access classes (How to maintain information or best way to interact with a user)

Importance of Good Design

- Time spent on design decides the success of the software developed.
- Good design simplifies implementation and maintenance of a project.
- To formalize (celebrate, honor) design process, axiomatic (clear) approach is to be followed


OO Design
Design, classes, methods, attributes and associations
Apply design axioms
Refine UML diagrams
Design view/ access layers and prototype
User satisfaction and usability tests based on use cases

10.3 ACTIVITIES OF OOD PROCESS

NOTES

1. Apply design axioms to design classes, their attributes, methods, associations, structure and protocols.

1.1 Refine and complete the static UML class diagram by adding details to the UML class diagram. These steps consist of the following activities:

1.1.1 Refine attributes

1.1.2 Design methods and protocols by utilizing a UML activity diagram to represent the methods algorithm.

1.1.3 Refine association between classes (if required)

1.1.4 Refine class hierarchy and design with inheritance (if required).

1.2 Iterate and refine again.

2. Design the access layer

2.1 Create mirror classes: For every business class identified and created, create one access layer

Eg , if there are 3 business classes (class1, class2 and class3), create 3 access layer classes (class1DB, class2DB and class3DB)

2.2 Identify access layer class relationships

2.3 Simplify classes and their relationships

– Main goal is to eliminate redundant classes and structures

2.3.1 Redundant classes: Do not keep two classes that perform similar translate request and translate results activities. Select one and eliminate the other.

2.3.2 Method classes: Revisit the classes that consist of only one or two methods to see if they can be eliminated or combined with existing classes.

2.4 Iterate and refine again.

3. Design the view layer classes

3.1 Design the macro level user interface, identifying view layer objects

3.2 Design the micro level user interface, which includes these activities:

3.2.1 Design the view layer objects by applying the design axioms and corollaries.

3.2.2 Build a prototype of the view layer interface

3.3 Test usability and user satisfaction

3.4 Iterate and refine

4. Iterate and refine the whole design. Reapply the design axioms and if needed, repeat the preceding steps.

10.4 OBJECT ORIENTED DESIGN AXIOMS

An axiom is a fundamental truth that always is observed to be valid and for which there is no counterexample or exception.

• They cannot be proven or derived but they can be invalidated by counter examples or exceptions.

• A theorem is a proposition that may not be self-evident (clear) but can be proved from accepted axioms.

• A corollary is a proposition that follows from an axiom or another proposition that has been proven.

Types of Axioms

• AXIOM-1 (Independence axiom): deals with relationships between systems
components such as classes, requirements, software components.

• AXIOM-2 (Information axiom): deals with the complexity of design

10.5 AXIOMS OF OOD

• The axiom 1 of object-oriented design deals with relationships between system components (such as classes, requirements and software components) and axiom 2 deals with the complexity of design.

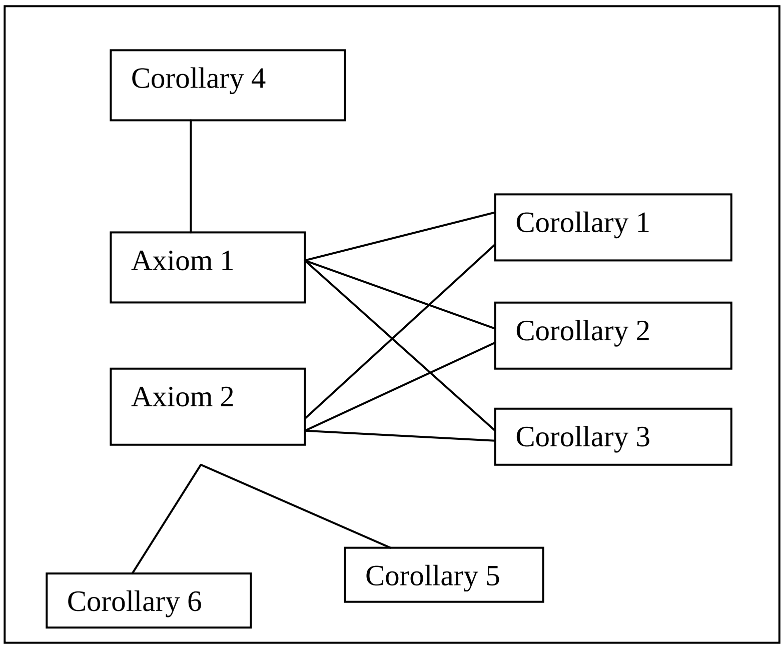
• Axiom 1 The independence axiom: Maintain the independence of components. According to axiom 1, each component must satisfy its requirements without affecting other requirements. Eg. Let us design a refrigerator door which can provide access to food and the energy lost should be minimized when the door is opened and closed. Opening the door should be independent of losing energy.

• Axiom 2 The information axiom: Minimize the information content of the design. It is concerned with simplicity. In object-oriented system, to minimize complexity use inheritance and the system's built in classes and add as little as possible to what already is there.

Check Your Progress

1. Define method classes.
2. Define axioms.
3. What is the purpose of axiom2?

10.6 COROLLARIES



NOTES

10.6.1 Types of corollaries

The design rules or corollaries derived from design axioms are stated below.

• Corollary 1: Uncoupled design with less information content.
• Corollary 2: Single purpose.
• Corollary 3: Large number of simple classes.
• Corollary 4: Strong mapping.
• Corollary 5: Standardization.
• Corollary 6: Design with inheritance.

Coupling

• Coupling is a measure of the strength of association established by a connection from one object or software component to another.
• Coupling is a binary relationship.
• For example A is coupled with B.
• Coupling is important when evaluating a design because it helps us focus on an important issue in design.

The degree of coupling

• How complicated the connection is ?
• Whether the connection refers to the object itself or something inside it.
• What is being sent or received?

Cohesion

• Cohesion can be defined as the interactions within a single object or software component.

Corollaries

- Corollary 1: Uncoupled design with less information content. Highly cohesive (interconnected) objects can improve coupling because only a minimal amount of essential information need be passed between objects.
- Corollary 2: Single purpose. Each class must have a single, clearly defined purpose. While documenting, one should be able to describe the purpose of a class in few sentences.
- Corollary 3: Large number of simple classes. Keeping the classes simple allows reusability.
- Corollary 4: Strong mapping. There must be a strong association between the physical system (analysis's objects) and logical design (design's object).
- Corollary 5: Standardization. Promote standardization by designing inter changeable components and reusing existing classes or components.
- Corollary 6: Design with inheritance. Common behavior (methods) must be moved to super classes. The superclass-subclass structure must make logical sense.

Corollary 1: Uncoupled Design with Less Information Content

- Coupling is a measure of the strength of association established by a connection from one object or software component to another. Coupling is a binary relationship. It is important for design because a change in one component should have a minimal impact on the other components.
- The degree or strength of coupling between 2 components is measured by the amount and complexity of information transmitted between them.
- Object oriented design has 2 types of coupling: interaction coupling and inheritance coupling.
- Interaction coupling involves the amount and complexity of messages between components. It is good to have little interaction. The general guideline is to keep the message as simple and infrequent as possible.
- Objects connected to many complex messages are tightly coupled, meaning any change to one invariably leads to a ripple (current, flow) effect of changes in others.
- Inheritance is a form of coupling between super and sub classes.
- A subclass is coupled to its superclass in terms of attributes and methods. We need high inheritance coupling. For this each specialization class should not inherit lot of unrelated and unneeded methods and attributes.
- If the superclass is overwriting most of the methods or not using them, then it is an indication that the inheritance coupling is low.

Types of coupling among objects or components (highest to lowest)

Name	Degree of coupling
• Content coupling	Very high
• Common coupling	High
• Control coupling	Medium
• Stamp coupling	Low
• Data coupling	Very low

Cohesion: The interactions within a single object or software component are called cohesion.

- Cohesion reflects the “single-purposeness” of an object.
- Highly cohesive components can lower coupling because only a minimum of essential information need be passed between components.
- Method cohesion means that a method should carry one function.
- A method that carries multiple functions is undesirable.
- Class cohesion means that all the class’s methods and attributes must be highly cohesive, meaning to be used by internal methods or derived classes’ methods.

Corollary 2: Single Purpose

- Every class should be clearly defined and necessary in the context of achieving the system's goals.
- When we document a class, we should be able to explain its purpose in a sentence or two.
- If we cannot, then the class should be subdivided into independent pieces.
- Each method must provide only one service.
- Each method should be of moderate size, no more than a page; half a page is better.

Corollary 3: Large number of simpler classes, Reusability

- There are benefits in having a large number of simpler classes because the chances of reusing smaller classes in other projects are high.
- Large and complex classes are too specialized to be reused.
- Object-oriented design offers a path for producing libraries of reusable parts.

Why reusability is not used? Coad and Yourdan, Software engineering textbooks teach new practitioners to build systems from "first principles"; reusability is not promoted or even discussed

- The "not invented here" syndrome (condition, pattern) and the intellectual (logical) challenge of solving an interesting software problem in one's own unique way mitigates against reusing someone else's software component.
- Unsuccessful experiences with software reusability in the past have convinced many practitioners and development managers that the concept is not practical.
- Most organizations provide no reward for reusability; sometimes productivity (output, efficiecy) is measured in terms of new lines of code written plus a discounted credit

Corollary 4: Strong mapping

- A strong mapping links classes identified during analysis and classes designed during the design phase eg view and access classes.
- The analyst identifies objects' types and inheritance, and thinks about events that change the state of objects.
- The designer adds detail to this model perhaps designing screens, user interaction, and client-server interaction.

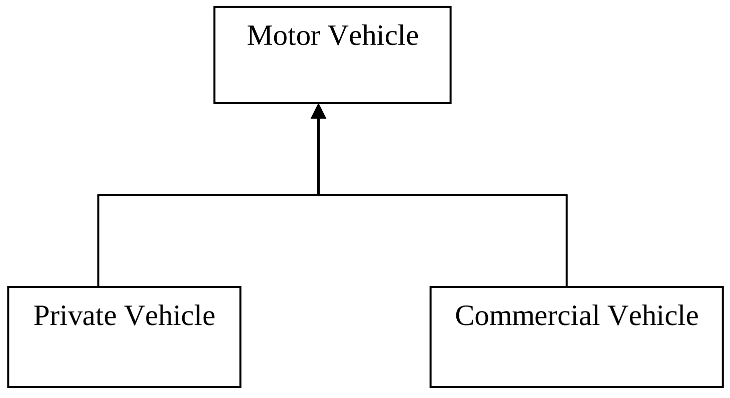
Corollary 5: Standardization

- To reuse classes, we must have a good understanding of the classes.

- Most object-oriented systems come with several built-in class libraries.
- But these class libraries are not always well documented.
- Sometimes they are documented, but not updated.
- They must be easily searched, based on users' criteria.

Corollary 6: Designing with inheritance

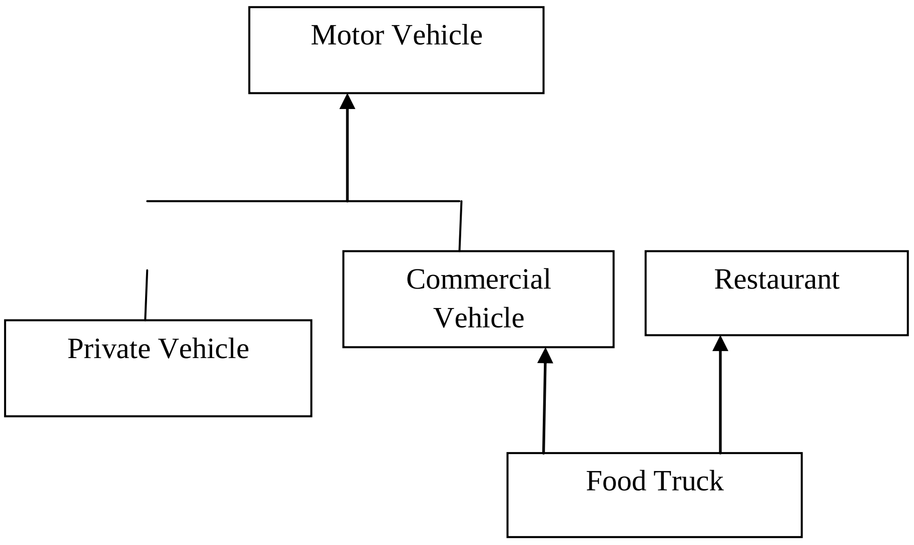
- When we implement a class, we have to determine its ancestor, what attributes it will have, and what messages it will understand.
- Then we have to construct its methods and protocols.
- Ideally, one has to choose inheritance to minimize the amount of program instructions.
- The primitive form of reuse is cut-and-paste reusability. Achieving Multiple inheritance in a Singe Inheritance System
- Single inheritance means that each class has only a single super class.
- The result of using a single inheritance hierarchy is the absence of ambiguity as to how an object will respond in a given method;
- We simply trace up the class tree beginning with the object's class, looking for a method of the same name.
- But languages like LISP or C++ have a multiple inheritance scheme whereby objects can inherit behaviour from unrelated areas of the class tree.
- The complication here is how to determine which behaviour to get from which class, particularly when several ancestors define the same method.
- One way of resolving this is to inherit from the most appropriate class and add an object of mother class as an attribute or aggregation. The other is to use the instance of the class (object) as an attribute.



Single Inheritance

NOTES

Multiple Inheritance



10.7 DESIGN PATTERNS

- A design pattern provides a scheme for refining the subsystem or components of a software system or the relationships among them.
- They allow systems to share knowledge about their design, by describing commonly recurring structures of communicating components that solve a general design problem within a particular context

10.8 INTRODUCTION TO DESIGNING CLASSES

Most important activity in designing an application is coming up with a set of classes that work together to provide the needed functionality. Underlying the functionality of any application is the quality of its design. OCL (Object Constraint Language- a specification language, provided by UML) specifies the properties of a system.

- Object-oriented design requires taking the object identified during object oriented analysis and designing classes to represent them.
- As a class designer, we have to know the specifics of the class we are designing and also we should be aware of how that class interacts with other classes.

10.9 OBJECT ORIENTED DESIGN PHILOSOPHY

- Here in terms of classes, as new facts are acquired, we relate them to existing structures in our environment (model).
- After enough new facts are acquired about a certain area, we create new structures to accommodate the greater level of detail in our knowledge.

- The important activity in designing an application is coming up with a set of classes that work together to provide the functionality we desire.
- If we design the classes with reusability in mind, we will gain a lot of productivity and reduce the time for developing new applications.

10.10 DESIGNING CLASSES: THE PROCESS

1. Apply design axioms to design classes, their attributes, methods, associations, structure and protocols.

   1.1 Refine and complete the static UML class diagram by adding details to that diagram.

   1.1.1 Refine attributes

   1.1.2 Design methods and the protocols by utilizing a UML activity diagram to represent the methods algorithm.

   1.1.3 Refine the associations between classes

   1.1.4 Refine the class hierarchy and design with inheritance

1.2 Iterate and Refine

10.11CLASS VISIBILITY: DESIGNING WELL-DEFINED PUBLIC, PRIVATE AND PROTECTED PROTOCOLS

- In designing methods or attributes for classes, we are confronted (deal) with two problems.
- One is the protocol or interface to the class operations and its visibility and the other is how it is implemented.
- The class’s protocol or the messages that a class understands, can be hidden from other objects (private protocol) or made available to other objects (public protocol).
- Public protocols define the functionality and external messages of an object.
- Private protocols define the implementation of an object.

Visibility

- A class might have a set of methods that it uses only internally, messages to itself. This private protocol of the class, includes messages that normally should not be sent from other objects. Here only the class itself can use the methods.
- The public protocol defines the stated behavior of the class as a citizen in a population and is important information for users as well as future descendants, so it is accessible to all classes. If the

NOTES

methods or attributes can be used by the class itself (or its subclasses) a protected protocol can be used. Here subclasses can used the method in addition to the class itself.

- The lack of well-designed protocol can manifest itself as encapsulation leakage. It happens when details about a class's internal implementation are disclosed through the interface.

Encapsulation leakage

- Encapsulation leakage is lack of a well-designed protocol
- The problem of encapsulation leakage occurs when details about a class's internal implementation are disclosed through the interface.
- As more internal details become visible, the flexibility to make changes in the future decreases.

Visibility types

- In UML the following types are used to specify export control
- + Public,
- # Protected
- Private

Check Your Progress

4. Define coupling.
5. What is cohesion?
6. Define design patterns.
7. Expand OCL and explain the purpose of it.
8. When an encapsulation leakage occurs explain?
9. List the types of visibility.

10.12 DESIGNING CLASSES: REFINING ATTRIBUTES

- Attributes identified in object-oriented analysis must be refined with an eye on implementation during this phase.
- In the analysis phase, the name of the attribute is enough.
- But in the design phase, detailed information must be added to the model.
- The 3 basic types of attributes are: (1) Single-value attributes (2) Multiplicity or multivalue attributes (3) Reference to another object or instance connection

Attributes

- Attributes represent the state of an object.

- When the state of the object changes, these changes are reflected in the value of attributes.
- Single value attribute has only one value or state. (Eg). Name, address, salary
- Multiplicity or multivalue attribute can have a collection of many values at any time.
- (Eg) If we want to keep tact of the names of people who have called a customer support line for help, we use multi value attribute
- Instance connection attributes are required to provide the mapping needed by an object to fulfill its responsibilities.
- (E.g.) A person may have one or more bank accounts.
- A person has zero to many instance connections to Account(s).
- Similarly, an Account can be assigned to one or more person(s) (joint account).
- So an Account has zero to many instance connection to Person(s).

10.13 UML ATTRIBUTE PRESENTATION

- During design, OCL (Object Constraint Language) can be used to define the class attributes

OCL

- The rules and semantics of the UML are expressed in English, in a form known as object constraint language.
- Object constraint language (OCL) is a specification language that uses simple logic for specifying the properties of a system.

The following is the attribute presentation suggested by UML:

Visibility name: type-expression=initial-value

Where visibility is one of the following:

+ public visibility (Accessibility to all classes)

# protected visibility (Accessibility to subclasses and operations of the class)

- private visibility (Accessibility only to operations of the class)

10.14 DESIGNING METHODS AND PROTOCOLS

- A class can provide several types of methods:
- Constructor: Method that creates instances (objects) of the class
- Destructor: The method that destroys instances
- Conversion Method: The method that converts a value from one unit of measure to another.

- Copy Method: The method that copies the contents of one instance to another instance
- Attribute set: The method that sets the values of one or more attributes
- Attribute get: The method that returns the values of one or more attributes
- I/O methods: The methods that provide or receive data to or from a device
- Domain specific: The method specific to the application.

Design goals

- To maximize cohesiveness (interconnection) among objects and software components to improve coupling (combination, pairing), because only a minimal amount of essential information should be passed between components.
- Abstraction leads to simplicity and straight- forwardness and, at the same time, increases class versatility (flexibility, resourcefulness, usefulness).

Five rules/ characteristics of bad design

- If it looks messy (confused, disorder), then it's probably a bad design.
- If it is too complex, then it's probably a bad design.
- If it is too big, then it's probably a bad design.
- If people don't like it, then it's probably a bad design.
- If it doesn't work, then it's probably a bad design.

10.15 DESIGN ISSUES- AVOIDING DESIGN PITFALL (DRAWBACKS, DIFFICULTY)

- Apply design axioms to avoid common design problems and pitfalls.
- Much better to have a large set of simple classes than a few large, complex classes.

Possible actions to solve design problems

- Keep a careful eye on the class design and make sure that an object's role remains well defined. If an object loses focus, you need to modify the design. Apply corollary-2
- Move some functions into new classes that the object would use, Apply corollary 1. • Break up the class into two or more classes. Apply corollary 3.
- Rethink the class definition based on experience gained.

NOTES

10.16 PACKAGES AND MANAGING CLASSES

- A package groups and manages the modeling elements, such as classes, their associations and their structures.
- Packages themselves may be nested within other packages.
- A package may contain both other packages and ordinary model elements.
- A package provides a hierarchy of different system components and can reference other packages.
- Classes can be packaged based on the services they provide or grouped into the business classes, access classes and view classes.

Check Your Progress

10. What are the types of attributes?
11. What is a constructor?
12. Define package.

10.17 ANSWERS TO CHECK YOUR PROGRESS QUESTIONS

1. Method classes revisit the classes that consist of only one or two methods to see if they can be eliminated or combined with existing classes.

2. An axiom is a fundamental truth that always is observed to be valid and for which there is no counterexample or exception.

3. Axiom2 minimizes the information content of the design. It is concerned with simplicity.

4. Coupling is a measure of the strength of association established by a connection from one object or software component to another.

5. Cohesion can be defined as the interactions within a single object or software component.

6. A design pattern provides a scheme for refining the subsystem or components of a software system or the relationships among them.

7. OCL (Object Constraint Language- a specification language, provided by UML) specifies the properties of a system.

8. The problem of encapsulation leakage occurs when details about a class's internal implementation are disclosed through the interface.

NOTES

9. The visibility types are Public, Protected, Private.

10. The 3 basic types of attributes are: (1) Single-value attributes (2) Multiplicity or multivalue attributes (3) Reference to another object or instance connection.

11. Constructor is a method that creates instances (objects) of the class.

12. A package groups and manages the modeling elements, such as classes, their associations and their structures.

10.17 SUMMARY

- The basic goal of the axiomatic approach is to formalize the design process and assist in establishing a scientific foundation for the object-oriented design process, so as to provide a fundamental basis for the creation of systems.
- Without scientific principles, the design field never will be systematized and so will be difficult to comprehend, codify, teach and practice.
- The main activities in design process are ◊ Designing classes (their attributes, methods, associations, structures, and protocols) and applying design axioms. ◊ Designing the access layer. ◊ Designing the user interface (view layer classes). ◊ Testing user satisfaction and usability, based on the usage and use cases. ◊ Iterating and refining the design.
- An axiom is a fundamental truth that always is observed to be valid and for which there is no counterexample or exception.
- The axioms cannot be proven or derived but they cannot be invalidated by counterexamples or exceptions.
- Axiom 1 deals with relationships between system components and Axiom 2 deals with the complexity of design.
- A corollary is a proposition that follows from an axiom or another proposition that has been proven.
- A corollary is shown to be valid if its referent axioms and deductive steps are valid.
- Corollary 1: Uncoupled design with less information content.
- Corollary 2: Single purpose.
- Corollary 3: Large number of simple classes.
- Corollary 4: Strong mapping.
- Corollary 5: Standardization.
- Corollary 6: Design with inheritance.

- Private protocol of the class includes messages that normally should not be sent from other objects. The messages are accessible only to operations of that class. Only the class itself can use the method.
- The public protocol defines the stated behaviour of the class so that it is accessible to all classes.
- In a protected protocol, subclasses can use the method in addition to the class itself.

10.18 KEYWORDS

- Design Axioms
- Corollary
- Coupling
- Cohesion
- Mapping
- Visibility
- OCL

10.19 REVIEW QUESTIONS

1. What is the task of design?
2. What is the significance of Occam's razor?
3. Define axiom.
4. Define theorem and corollary.
5. What is coupling?
6. State the different type of coupling.
7. Define cohesion.
8. What is basic activity in designing an application?
9. List the Object Oriented design axioms and corollaries.
10. What is the relationship between coupling and cohesion?
11. What are public and private protocols? What is the significance of separating these two protocols?
12. What are some characteristics of a bad design?
13. How do design axioms help avoid design pitfalls?
14. List out the type of attributes. Explain them
15. How is an attribute represented in the UML?
16. How is an operation represented in the UML?
17. Define Encapsulation Leakage
18. What is OCL?
19. What is the use of protected protocol?

NOTES

10.20 FURTHER READINGS

1. Bahrami, A.(1999). Object Oriented Systems Development, Using the unified modeling language, McGraw-Hill

2. Object Oriented Analysis and Design using UML, by Rational Software Corporation (2002)

UNIT XI – DESIGNING METHODS

NOTES

Structure

NOTES

11.0 INTRODUCTION

- DBMS = is a set of programs that enables the creation and maintenance of a collection of related data provide a reliable, persistent
- Fundamental purpose of a DBMS data storage facility and the mechanisms for efficient, convenient data access and retrieval
- In an OO system, it concerns both persistent objects and transient objects.
- Transient data - data that will be erased from memory after they have been used

Persistent data system, data that must be stored in secondary data storage not just in computer memory, that must be stored after the program that creates or amends it stops running, and that usually must be available to other users

11.1 OBJECTIVES

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

- Understand about access layer
- Understand Database model
- Describe Database interface
- Understand about client server computing & distributed databases
- Understand about CORBA
- Understand Object oriented database system
- Describe view layer

11.2 ACCESS LAYER: OBJECT STORAGE AND OBJECT INTEROPERABILITY

- A Date Base Management System (DBMS) is a set of programs that enables the creation and maintenance (access, manipulate, protect and manage) of a collection of related data.
- The purpose of DBMS is to provide reliable, persistent data storage and mechanisms for efficient, convenient data access and retrieval.
- Persistence refers to the ability of some objects to outlive the programs that created them.
- Object lifetimes can be short for local objects (called transient objects) or long for objects stored indefinitely in a database (called persistent objects). Persistent stores
- Most object-oriented languages do not support serialization or object persistence, which is the process of writing or reading an object to and from a persistence storage medium, such as disk file.

- Unlike object oriented DBMS systems, the persistent object stores do not support query or interactive user interface facilities.
- Controlling concurrent access by users, providing ad-hoc query capability and allowing independent control over the physical location of data are not possible with persistent objects.

11.2.1 Object Store and Persistence

Each item of data will have a different lifetime. These lifetimes are categories into six, namely

1. Transient results to the evaluation of expressions.
2. Variable involved in procedure activation (parameters and variables with a localized scope)
3. Global variable and variables that are dynamically allocated
4. Data that exist between the executions of a program
5. Data that exist between the versions of a program.
6. Data that outlive a program.

Typically, programming languages provide excellent, integrated support for the first three categories of transient data. The other three categories can be supported by a DBMS, or a file system. A file or a database can provide a longer life for objects – longer than the duration of the process in which they were created. From a language perspective, this characteristic is called persistence.

Essential elements in providing a persistent store are:

¾ Identification of persistent objects or reachability (Object ID)

¾ Properties of objects and their interconnections. The store must be able to coherently manage nonpointer and pointer data. (i.e interobject references)

¾ Scale of the object store. The object store should provide a conceptually infinite store.

¾ The system should be able to recover from unexpected failures and return the system to a recent self-consistent state.

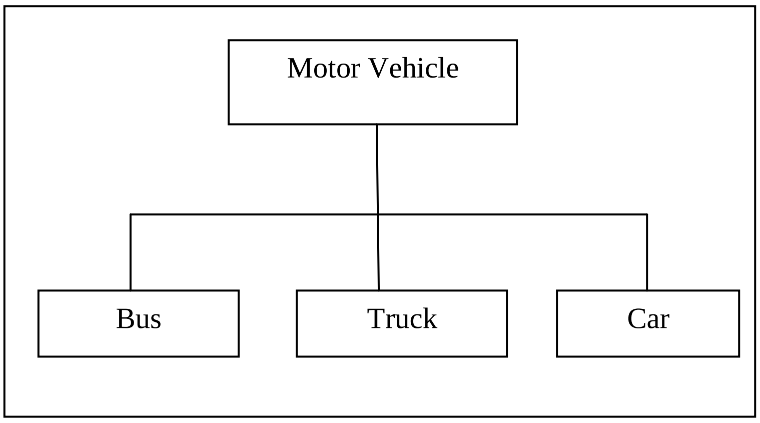
11.3 DATABASE MODELS

A database model is a collection of logical constructs used to represent the data structure and data relationships within the database. The conceptual model focuses on the logical nature of that data presentation. It is concerned with what is represented in the database and the implementation model is concerned with how it is represented.

11.3.1 Hierarchical model

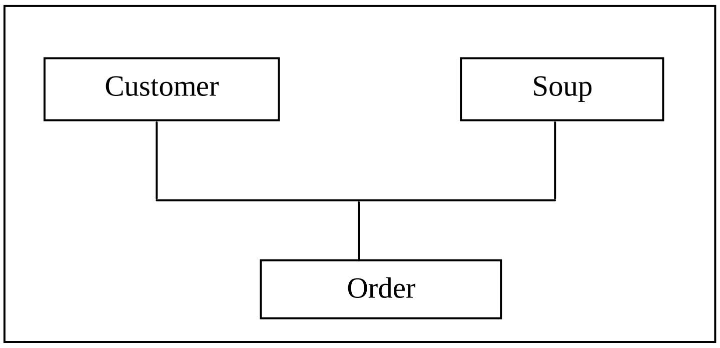
This model represents data as a single rooted tree. Each node in the tree represents a data object and the connection represents a parent-child relationship.

NOTES



11.3.2 Network Model

This is similar to a hierarchical database, with one difference. Here record can have more than one parent. Each parent can have any number of child nodes and each child node can have any number of parent nodes.



11.3.3 Relational Model

- It is simple and widespread. Here the relation can be thought of as a table.
- The columns of each table are attributes that define the data or value domain for entries in that column.
- The rows of each table are tuples representing individual data objects being stored.
- A relational table should have only one primary key.
- A primary key is a combination of one or more attributes whose value unambiguously (clearly) locates each row in the table.
- A foreign key is a primary key of one table that is embedded in another table to link the tables.

11.4 DATABASE INTERFACE

- The interface on a database must include a data definition language (DDL), a query and data manipulation language (DML). These languages must be designed to reflect the flexibility and constraints inherent in the data model. Databases have adopted 2 approaches for interfaces with the system.

- One is to embed a database language such as structured query language (SQL), in the host programming language. The problem here is that application programmers have to learn and use two different languages. The application programmers have to negotiate the differences in the data models and data structures allowed in both languages.
- Another approach is to extend the host programming language with database related constructs. Here application programmers need to learn only a new construct of the same language rather than a completely new language. Eg. GemStone from Servio Logic has extended the Smalltalk object-oriented programming.

11.4.1 Database Schema and Data Definition Language

- DDL is the language used to describe the structure of and relationships between objects stored in a database. This structure of information is termed the database schema. In traditional databases, the schema is the collection of record types and set types or the collection of relationships and table records to store information about entities of interest to the application.
- E.g.. CREATE TABLE inventory (Inventory_Number CHAR(10) NOT NULL Description CHAR(25) NOT NULL Price DECIMAL (9,2));

11.4.2 Data Manipulation Language and Query Capabilities

- Asking Questions, formally making queries of the data – is a typical and common use of a database. A query is expressed through a query language. A Data Manipulation Language (DML) is the language that allows users to access and manipulate (such as create, save or destroy) data organizations. The SQL is the standard DML for relational DBMSs. The query usually specifies
- The domain of the discourse over which to ask the query
- The elements of general interest
- The conditions or constraints that apply
- The ordering, sorting or grouping of elements and the constraints that apply to the ordering or grouping

DML

- Query processes have sophisticated "engines" that determine the best way to approach the database and execute the query over it. They may use the information in the database or knowledge of the whereabouts of particular data in the network to optimize the retrieval of a query.
- DML are either procedural or nonprocedural. A Procedural DML requires users to specify what data are desired and how to get the data.

NOTES

- A non-procedural DML requires users to specify what data are needed but not how to get the data. Object-oriented query and data manipulation languages, such as Object SQL, provide object management capabilities to the data manipulation language.
- In a relational DBMS, the DML is independent of the host programming language.
- A host language such as C or COBOL would be used to write the body of the application. SQL statements then are embedded in C or COBOL applications to manipulate data.
- Once SQL is used to request and retrieve database data, the results of the SQL retrieval must be transformed into the data structures of the programming language.
- The drawback here is that the programmers code here in two languages, SQL and the host language.

Check Your Progress

1. Define DBMS.
2. Give the purpose of DBMS.
3. Define database model.
4. Define primary key
5. Define DDL
6. Define DML

11.5 LOGICAL AND PHYSICAL DATABASE ORGANIZATION AND ACCESS CONTROL

Logical database organization refers to the conceptual view of database structure and the relationships within the database. Physical database organization refers to how the logical components are represented in a physical form by operating system constructs eg objects may be represented as files.

11.5.1 Shareability and Transactions

Data and objects in the database need to be accessed and shared by different applications. With multiple applications having access to the object concurrently, it is likely that conflicts over object access will arise. The database must detect and mediate these conflicts and promote maximum amount of sharing without any data integrity problem. This mediation process is managed through concurrency control policies, implemented, by transactions.

11.5.2 Transactions

Transaction is a unit of change in which many individual modifications are aggregated into a single modification that occurs in its entirety or not at all. Thus either all changes to objects within a given transaction are applied to the database or none of the changes. A transaction is said to commit if all changes can be made successfully to the database and to abort if cancelled because all change to the database cannot be made successfully. This ability of transactions ensures atomicity of change that maintains the database in a consistent state.

Many transaction systems are designed for short transactions (lasting for minutes). They are less suitable for long transactions, lasting hours. Object databases are designed to support both short and long transactions. A concurrent control policy dictates what happens when conflicts arise between transactions that attempt to access to the same object and how these conflicts are to be resolved.

11.5.3 Concurrency Policy

When several users attempt to read and write the same object simultaneously, they create a contention (conflict) for object. Then concurrency control mechanism is established to mediate such conflicts by making policies that dictate how they will be handled.

To provide consistent view, the transactions must occur in serial order. A user must see the database as it exists before a given transaction occurs or after the transaction.

11.5.4 Concurrency issues

- The conservative way of enforcing serialization is to allow a user to lock all objects or records when they are accessed and to release the locks only after a transaction commits. This is called conservative or pessimistic policy. It provides exclusive access to the object, despite what is done to it. The policy is conservative because no other user can view the data until the object is released.
- By distinguishing between querying (reading) the object and writing to it, greater concurrency can be achieved. This policy allows many readers of an object but only one writer.
- Under an optimistic policy, two conflicting transactions are compared in their entirety and then their serial ordering is determined.
- A process can be allowed to obtain a read lock on an object already write locked if its entire transaction can be serialized as if it occurred either entirely before or entirely after the conflicting transaction.
- The reverse also is true. A process may be allowed to obtain a write lock on an object that has a read lock if its entire transaction can be serialized as if it occurred after the conflicting transaction. Now the optimistic policy allows more processes to operate concurrently than the conservative policy.

11.6 DISTRIBUTED DATABASES AND CLIENT-SERVER COMPUTING

- In distributed databases, portions of the database reside on different nodes and disk drives in the network.

- Each portion of the database is managed by a server.
- The server sends information to client applications and makes queries or data requests to these client applications or other servers.

11.6.1 Client-Server Computing

- It is the logical extension of modular programming. In modular programming we separate a large piece of software into its constituent parts (modules). This makes development easier and gives better maintainability.
- In client-server computing all those modules are not executed within the same memory space or even on the same machine. Here the calling method becomes “client” and the called module becomes the “server”.
- The important component of client-server computing is connectivity, which allows applications to communicate transparently with other programs or processes, regardless of their locations. The key element of connectivity is the network operating system (NOS), known as middleware. The NOS provides services such as routing, distribution, messages, filing, printing and network management.
- Client programs manage user interface portion of the application, validate data entered by the user, dispatch requests to server program and executes business logic. The business layer contains all the objects that represent the business.
- The client-based process is the front-end of the application, which the user sees and interacts with. It manages the local resource with which the user interacts, such as the monitor, keyboard, workstation, CPU and peripherals.
- A key component of a client workstation is the graphical user interface (GUI). It is responsible for detecting user actions, managing the Windows on the display and displaying the data in the Windows.
- The server process performs back-end tasks.
- Client – Node that request for a service
- Server – Node that services the request.
- Client Server computing is the logical extension of modular programming.
- The fundamental concept behind the modular programming is decomposing the larger software in to smaller modules for easier development and maintainability.
- Client Server computing is developed by extending this concept i.e, modules are allowed to execute in different nodes with different memory spaces.

- The module that needs and request the service is called a client and the module that gives the service is called a server.
- The network operating system is the back bones of this client server computing.
- It provides services such as routing, distribution, messages, filing and printing and network management. This Network Operating System (NOS) is called middleware.

Client Program:

- It sends a message to the server requesting a service (task done by server).
- Manages User Interface portion of the application.
- Performs validation of data input by the user.
- Performs business logic execution (in case of 2 tier).
- Manages local resources.
- Mostly client programs are GUI.
- Server Program:
- Fulfills the task requested by the client.
- Executes database retrieval and updation as requested by the client.
- Manages data integrity and dispatches results to the client.
- Some cases a server performs file sharing as well as application services.
- Uses power full processors and huge storage devices.
- File Server – Manages sharing of files or file records. Client sends a message to the file server requesting a file or file record. The File Server checks the integrity and availability of file/record.
- Data Base Servers – Client pass the SQL query in the form of messages to the server in turn server performs the query and dispatches the result.
- Transaction Servers – Client sends message to the server for a transaction (set of SQLstatements) where the transaction succeeds or fails entirely.
- Application Servers – Application servers need not to be database centric. They may serve any of user needs such as sending mails, regulating download.

Characteristics of Client Server Computing

1. A combination of client/ front end process that interacts with the user and server/ backend process that interacts with the shared resources.

2. The front end and back end task have different computing resource requirements.
3. The hardware platform and operating system need not be the same.
4. Client and Server communicate through standard well defined Application Program Interface(API).
5. They are scalable.

File server Vs Database server

- The server can take different forms. The simplest form of server is a file server. With a file server, the client passes requests for files or file records over a network to the file server. This needs a large bandwidth and can slow down a network with many users. Traditional LAN computing allows users to share resources such as data files and peripheral devices.
- Advanced forms of servers are database servers, transaction servers, and application servers and object servers. With database servers, clients pass SQL requests as messages to the server and the results of the query are returned over the network. Both the code that processes the SQL request and the data reside on the server, allowing it to sue its own processing power to find the requested data. This is in contrast to the file server, which requires passing all the records back to the client and then letting the client find its own data.

Transaction Servers

- With transaction servers, clients invoke remote procedures that reside on servers, which also contain an SQL database engine. The server has procedural statements to execute a group of SQL statements (transactions), which either all succeed or fail as a unit.
- Applications based on transaction servers, handled by on-line transaction processing (OLTP), tend to be mission-critical applications that always require a 1-3 second response time and tight control over the security and integrity of the database. The communication overhead in this approach is kept to a minimum, since the exchange consists of a single request and reply (as opposed to multiple SQL statements in database servers).

N-tier architecture

- In a two-tier architecture, a client talks directly to a server, with no intervening server. This type of architecture is used in small environments with less than 50 users. To scale up to hundreds or thousands of users, it is necessary to move to a 3-tier architecture.
- Three-tier architecture introduces a server between the client and the server. The role of the application or Web server is manifold. It can provide translation services, metering services (transaction monitor to limit the number of simultaneous requests to a given server) or intelligent agent services (mapping a request to a

number of different servers, collating the results, and returning a single response to the client).

Basic characteristics of client-server architectures

- The client process contains solution-specific logic and provides the interface between the user and the rest of the application system. The server process acts as a software engine that manages shared resources such as databases, printers, modems or processors.
- The front end task and back-end task have fundamentally different requirements for computing resources such as processor speeds, memory, disk speeds and capacities and i/o devices.
- The environment is heterogeneous and multivendor. The h/w platform and o/s of client and server are not the same.
- They can be scaled horizontally and vertically. Horizontal scaling means adding or removing client workstations with only a slight performance impact. Vertical scaling means migrating to a larger and faster server machine.

11.7 DISTRIBUTED AND COOPERATIVE PROCESSING

- Distributed processing means distribution of applications and business logic across multiple processing platforms. It implies that processing will occur on more than one processor to complete a transaction. The processing is distributed across 2 or more machines, where each process performs part of an application in a sequence. These processes may not run at the same time.
- Cooperative processing is computing that requires two or more distinct processors to complete a single transaction. Here programs interact and execute concurrently on different processors.

11.8 COMMON OBJECT REQUEST BROKER ARCHITECTURE

- It is used to integrate distributed, heterogeneous business applications and data. The CORBA interface definition language (IDL) allows developers to specify language-neutral, object-oriented interfaces for application and system components. IDL definitions are stored in an interface repository that offers object interfaces and services. For distributed enterprise computing, the interface repository is central to communication among objects located on different systems.
- CORBA implements a communication channel through which applications can access object interfaces and request data and services.
- The CORBA common object environment (COE) provides system level services such as life cycle management for objects accessed

NOTES

through CORBA, event notification between objects and transaction and concurrency control.

11.9 MICROSOFT'S ACTIVEX/DCOM

- Microsoft's Component Object Model (COM) and its successor, the distributed components object model (DCOM) are alternative to COR BA. DCOM was bundled with Windows NT 4.0. DCOM is an Internet and component strategy where ActiveX (formerly known as object linking and embedding or OLE) plays the role of DCOM object. DCOM is backed by web browser

Check Your Progress

7. Define logical database organization.
8. Define client programs
9. Define application server
10. Define cooperative processing
11. Give the system level services offered by CORBA

11.10 OBJECT –ORIENTED DATABASE MANAGEMENT SYSTEMS

- It is marriage of object oriented programming and database technology. The defined operations apply universally and are not dependent on the particular database application running at the moment.
- The data types can be extended to support complex data such as multimedia by defining new object classes that have operations to support new kinds of information.
- It should have object-oriented language properties and database requirements.

11.10.1 Rules to make object-oriented system

- The system must support complex objects.
- Object identity must be supported
- Objects must be encapsulated
- System must support types or classes
- System must support inheritance
- System must avoid premature binding
- System must be computationally complete
- System must be extensible
- It must be persistent, able to remember an object state
- It must be able to manage large databases

- It must accept concurrent users
- Data query must be simple
- The system must support complex objects. System must provide simple atomic types of objects (integers, characters, etc) from which complex objects can be built by applying constructors to atomic objects.
- Object identity must be supported: A data object must have an identity and existence independent of its values.
- Objects must be encapsulated: An object must encapsulate both a program and its data.
- System must support types or classes: The system must support either the type concept or class concept
- System must support inheritance: Classes and types can participate in a class hierarchy. Inheritance factors out shared code and interfaces.
- System must avoid premature binding: This is also known as late binding or dynamic binding. It shows that the same method name can be used in different classes. The system must resolve conflicts in operation names at run time.
- System must be computationally complete: Any computable function should be expressible in DML of the system, allowing expression of any type of operation.
- System must be extensible: The user of the system should be able to create new types that have equal status to the system's predefined types.
- It must be persistent, able to remember an object state: System must allow the programmer to have data survive beyond the execution of the creating process for it to be reused in another process.
- It must be able to manage large databases: System must manage access to the secondary storage and provide performance features such as indexing, clustering, buffering and query optimization.
- It must accept concurrent users: System must allow multiple concurrent users and support notions of atomic, serializable transactions ,must be able to recover from hardware and software failures
- Data query must be simple: System must provide some high-level mechanism for ad-hoc browsing of the contents of the database.

11.11 OBJECT ORIENTED DATABASES VERSUS TRADITIONAL DATABASES

The responsibility of an OODBMS includes definition of the object structures, object manipulation and recovery, which is the ability to maintain data integrity regardless of system, network or media failure. The OODBMs like DBMSs must allow for sharing; secure concurrent multiuser access; and efficient, reliable system performance. The objects are an “active” component in an object-oriented database, in contrast to conventional database systems, where records play a passive role. Another feature of object-oriented database is inheritance. Relational databases do not explicitly provide inheritance of attributes and methods. The objects are an active component in an object-oriented database, in contrast to conventional database systems, where records play a passive role. The relational database systems do not explicitly provide inheritance of attributes and methods while object-oriented databases represent relationships explicitly (openly, clearly). (Improvement in data access performance) Object oriented databases also differ from the traditional relational databases in that they allow representation and storage of data in the form of objects. (Each object has its own identity or object-ID)

Object Identity

Object oriented databases allow representation and storage of data in the form of objects. Each object has its own identity or object-ID (as opposed to the purely value oriented approach of traditional databases). The object identity is independent of the state of the object.

11.12 DISTRIBUTED DATABASES

Many modern databases are distributed databases, which imply that portions of the database reside on different nodes (computers) and disk drives in the network. Usually, each portion of the database is managed by a server, a process responsible for controlling access and retrieval of data from the database portion.

Distributed and Cooperative Processing

Distributed Processing means distribution of applications and business logic across multiple processing platforms. Distributed processing implies that processing will occur on more than one processor in order for a transaction to be completed. In other words, processing is distributed across two or more machines, where each process performs part of an application in a sequence. Example, in processing an order from a client, the client information may process at one machine and the account information then may process on a different machine.

Cooperative processing is computing that requires two or more distinct processors to complete a single transaction. Cooperative processing is related to both distributed and client-server processing. Cooperative processing is a form of distributed computing in which two or more distinct processes are required to complete a single business transaction. Usually, these programs interact and execute concurrently on different processors. Cooperative processing also can be considered to be a style of distributed processing, if communication between processors is performed through a message passing architecture.

Distributed Object Computing

Distributed Object Computing (DOC) promises the most flexible client-server system, because it utilizes reusable software components that can roam anywhere on networks, run on different platforms, communicate with legacy applications by means of object wrappers.

Distributed objects are reusable software components that can be distributed and accessed by users across the network. These objects can be assembled into distributed applications. Distributed object computing introduces a higher level of abstraction into the world of distributed applications. Applications no longer consist of clients and servers but users, objects and methods. The user no longer needs to know which server process performs a given function. All information about the function is hidden inside the encapsulated object. A message requesting an operation is sent to the object, and the appropriate method is invoked.

Distributed object computing resulted from the need to integrated mission-critical applications and data residing on systems that are geographically remote, sometimes from users and often from each other, and running on many different hardware platforms. The business have had to integrate applications and data by writing custom interfaces between systems, forcing develops to spend their time building and maintaining an infrastructure rather than adding new business functionality.

11.13 OBJECT-RELATIONAL SYSTEMS

The object-oriented development creates a fundamental mismatch between the programming model (objects) and the way in which existing data are stored (relational tables).

To resolve the mismatch, a mapping tool between the application objects and the relational data must be established. Creating an object model from an existing relational database layout (schema) often is referred to as reverse engineering. Conversely, creating a relational schema from an existing object model often is referred to as forward engineering.

Tools that can be used to establish the object-relational mapping processes have begun to emerge. The main process in relational and object integration is defining the relationships between the table structures (represented as schemata) in the relational database with classes (representing classes) in the object model. Example Sun Java Blend allows the developer access to relational data as java objects, thus avoiding the mismatch between the relational and object data model.

11.14 OBJECT- RELATION MAPPING

In a relational database, the schema is made up of tables, consisting of rows and columns, where each column has a name and a simple data type. In an object model, a table is a class, which has a set of attributes (properties or data members). Object classes describe behaviour with methods.

A tuple (row) of a table contains data for a single entity that correlates to an object (instance of a class) in an object –oriented system. In addition, a stored procedure in a relation database may correlate to a method in an object-oriented architecture. A stored procedure is a module of precompiled SQL code maintained within the database that executes on the server to enforce rules the business has set about the data.

Therefore, the mapping essential to object and relational integration are between a table and a class, between columns and attributes, between a row and an object, and between a stored procedure and a method. The relational data maps to and from application objects, it must have at least the following mapping capabilities.

Table-class mapping

Table-multiple classes mapping

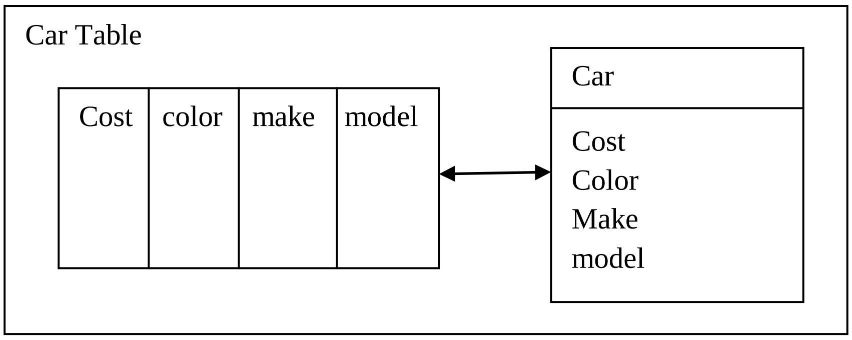
Table-inherited classes mapping

Tables- inherited classes mapping.

The tool must describe both how the foreign key can be used to navigate among classes and instances in the mapped object model and how referential integrity is maintained. Referential integrity means making sure that a dependent table's foreign key contains a value that refers to an existing valid tuple in another relation

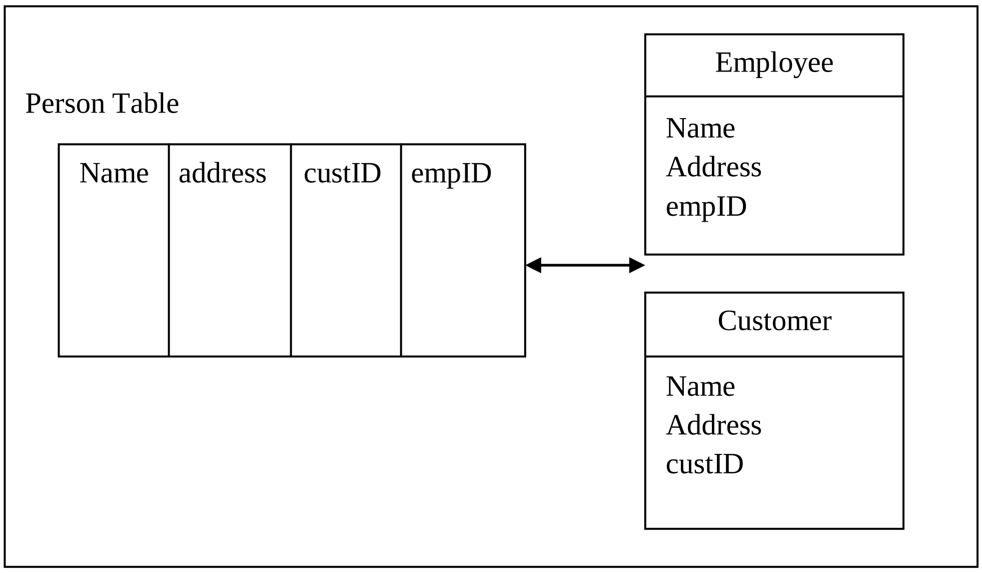
11.14.1 Table-class mapping

Table-Class mapping is a simple one-to-one mapping of a table to a class and the mapping of columns in a table to properties in a class. In this mapping, a single table is mapped to a single class.



11.14.2 Table-multiple classes mapping

In the table-multiple classes mapping, a single table maps to multiple non inheriting classes. Two or more distinct, non inheriting classes have properties that are mapped to columns in a single table. At run time, a mapped table row is accessed as an instance of one of the classes, based on a column value in the table.

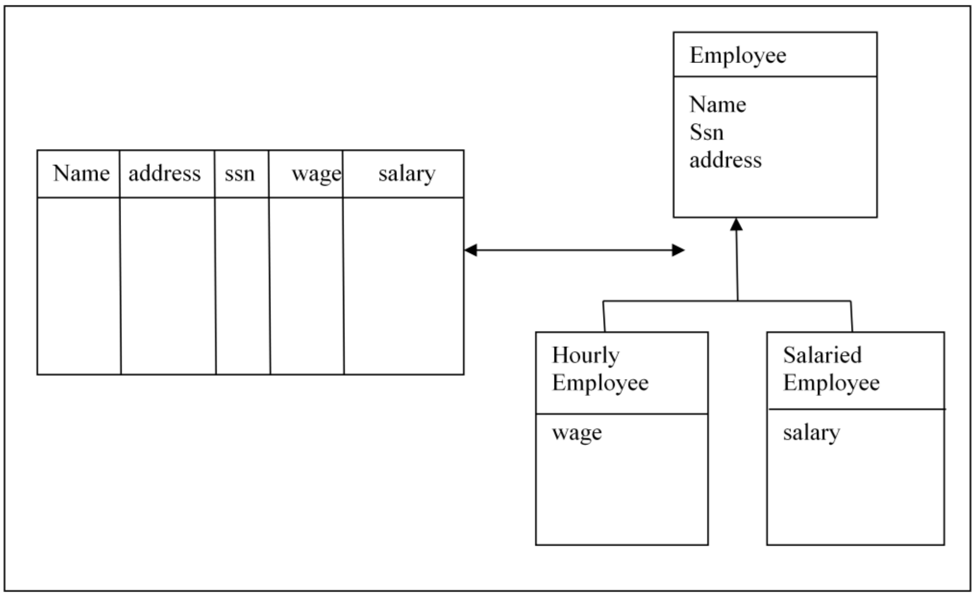


11.14.3 Table – Inherited Classes Mapping

In table-inherited classes mapping, a single table maps to many classes that have a common superclass. This mapping allows the user to specify the columns to be shared among the related classes. The superclass may be either abstract or instantiated.

Tables – Inherited classes Mapping

Another approach here is table-inherited classes mapping, which allows the translation of is a relationships that exist among tables in the relational schema into class inheritance relationships in the object model. In a relational database, an is-a relationship often is modeled by a primary key that acts as a foreign key to another table. In the object-model, is another term for an inheritance relationship

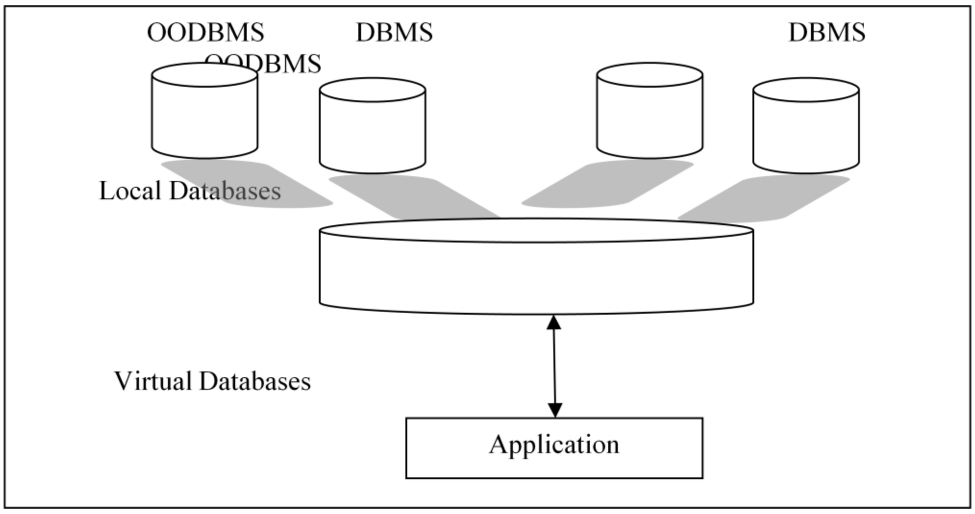


11.15 MULTI DATABASE SYSTEMS

- A different approach for integration object-oriented applications with relational data environments is multi database systems or heterogeneous database systems, which facilitate the integration of heterogeneous databases and other information sources.
- Heterogeneous information systems facilitate the integration of heterogeneous information sources, where they can be structured (having regular schema), semi-structured and sometimes even unstructured. Some heterogeneous information systems are constructed on a global schema over several databases. So users can have the benefits of a database with a schema to access data stored in different databases and cross database functionality. Such heterogeneous information systems are referred to as federated multidatabase systems.

Federated multidatabase systems provide uniform access to data stored in multiple databases that involve several different data models.

- A multidatabase system (MDBS) is a database system that resides unobtrusively on top of, existing relational and object databases and file systems (local database systems) and presents a single database illusion to its users.



- The MDBS maintains a single global database schema and local database systems maintain all user data.
- The schematic differences among local databases are handled by neutralization (homogenization), the process of consolidating the local schemata.
- The MDBS translates the global queries and updates for dispatch to the appropriate local database system for actual processing, merges the results from them and generates the final result for the user.
- MDBS coordinates the committing and aborting of global transactions by the local database systems that processed them to maintain the consistency of the data within the local databases.
- An MDBS controls multiple gateways (or drivers). It manages local databases through gateways, one gateway for each local database.
- Open Data Base Connective (ODBC) is an application programming interface that provides solutions to the multi database programming problem. It provides a vendor-neutral mechanism for independently accessing multiple database hosts.
- ODBC and other APIs provide standard database access through a common client-side interface. It avoids the burden of learning multiple database APIs. Here one can store data for various applications or data from different sources in any database and transparently access or combing the data on an as needed basis. Details of back-end data structure are hidden from the user.

11.15.1 ODBC

- ODBC is similar to Windows print model, where the application developer writes to a generic printer interface and a loadable driver maps that logic to hardware-specific commands.

- This approach virtualizes the target printer or DBMS because the person with the specialized knowledge to make the application logic work with the printer or database is the driver developer and not the application programmer.
- The application interacts with the ODBC driver manager, which sends the application calls (such as SQL statements) to the database.
- The driver manager loads and unloads drivers, perform status checks and manages multiple connections between applications and data sources.

Check Your Progress

12. What is the responsibility of an OODBMS?
13. Define distributed databases.
14. Define Distributed Object Computing.
15. Define tuple.
16. What is table class mapping?
17. Give the advantage of heterogeneous information.
18. Define ODBC.
19. What is the purpose of driver manager?

11.16 DESIGNING ACCESS LAYER CLASSES

- The main idea behind creating an access layer is to create a set of classes that know how to communicate with the place(s) where the data actually reside. Regardless of where the data reside, whether it be a file, relational database, mainframe, Internet, DCOM or via ORB, the access classes must be able to translate any data-related requests from the business layer into the appropriate protocol for data access.
- These classes also must be able to translate the data retrieved back into the appropriate business objects.
- The access layer's main responsibility is to provide a link between business or view objects and data storage.
- Three-layer architecture is similar to 3-tier architecture. The view layer corresponds to the client tier, the business layer to the application server tier and the access layer performs two major tasks:

Access Layer tasks

- Translate the request: The access layer must be able to translate any data related requests from the business layer into the appropriate protocol for data access.

- Translate the results: The access layer also must be able to translate the data retrieved back into the appropriate business objects and pass those objects back into the business layer.

11.16.1 Advantage of this approach

- Here design is tied to any base engine or distributed object technology such as CORBA or DCOM. Here we can switch easily from one database to another with no major changes to the user interface or business layer objects. All we need to change are the access classes' methods.
- The access layer design process consists of the following activities:
- If a class interacts with nonhuman actor such as another system, database or the web, then the class automatically should become an access class.

11.17 DESIGN THE ACCESS LAYER

(1) For every business class identified, mirror the business class package.: For every business class identified and created, create one access class in the access layer package. Eg , if there are 3 business classes (class1, class2 and class3), create 3 access layer classes (class1DB, class2DB and class3DB)

(2) Identify access layer class relationships (or) define the relationships.

- Simplify classes and their relationships – main goal is to eliminate redundant classes and structures
- Redundant classes: Do not keep 2 classes that perform similar translate request and translate results activities. Select one and eliminate the other.
- Method classes: Revisit the classes that consist of only one or two methods to see if they can be eliminated or combined with existing classes.
    - Iterate and refine again.

(3) Simplify classes and their relationships – main goal is to eliminate redundant classes and structures. In most cases , combine simple access class and simplify the super and subclass structures

Redundant classes: Do not keep 2 classes that perform similar translate request and translate results activities. Select one and eliminate the other.

Method classes: Revisit the classes that consist of only one or two methods to see if they can be eliminated or combined with existing classes.

(4) Iterate and refine again.

In this process, the access layer classes are assumed to store not only the attributes but also the methods. This can be done by utilizing an

OODBMS OR Relational data base Design the access layer process (approach-2)

• Let methods to be stored in a program (eg: a compiled c++ program stored on a file) and store not only the persistent attributes

(1) For every business class identified determine if class has persistent data. An attribute can be either transient or persistent (non transient)

(2) For every business class identified , mirror the business class package.: For every business class identified and created, create one access class in the access layer package. Eg , if there are 3 business classes (class1, class2 and class3), create 3 access layer classes (class1DB, class2DB and class3DB)

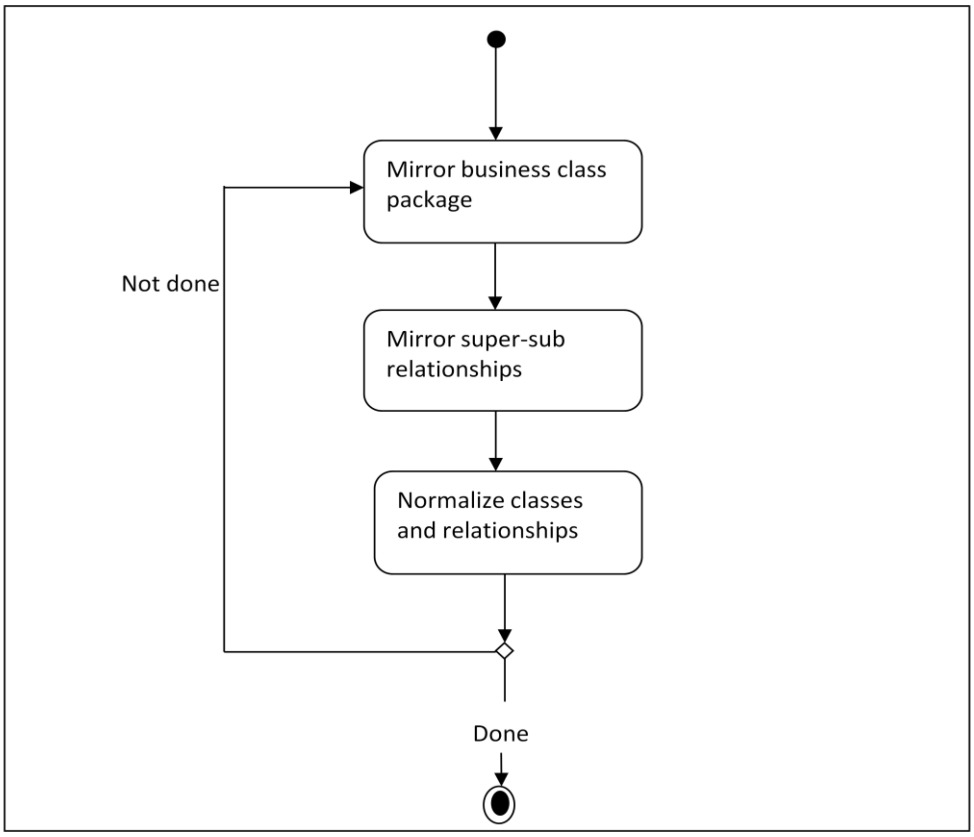
(3) Identify access layer class relationships (or) define the relationships.

(4) Simplify classes and their relationships – main goal is to eliminate redundant classes and structures. In most cases, combine simple access class and simplify the super and subclass structures

Redundant classes: Do not keep 2 classes that perform similar translate request and translate results activities. Select one and eliminate the other.

Method classes: Revisit the classes that consist of only one or two methods to see if they can be eliminated or combined with existing classes.

(5) Iterate and refine again.


Mirror business class package
Not done
Mirror super-sub relationships
Normalize classes and relationships
Done

Process of creating access layer classes

NOTES

11.18 INTRODUCTION TO VIEW LAYER

View layer objects are more responsible for user interaction and these view layer objects have more relation with the user where business layer objects have less interaction with users. Another feature of view layer objects are they deal less with the logic. They help the users to complete their task in an easy manner.

The Major responsibilities of view layer objects are

1. Input – View Layer objects have to respond for user interaction. The user interface is designed to translate an action by the user (Eg. Clicking the button) in to a corresponding message.

2. Output - Displaying or printing information after processing.

View Layer Design Process:

1. Macro Level UI Design Process

a. Identify classes that interact with human actors

b. A sequence/ collaboration diagram can be used to represent a clear picture of actor system interaction.

c. For every class identified determine if the class interacts with the human actor.

If so   i. Identify the view layer object for that class.

ii. Define the relationship among view layer objects.

2. Micro Level UI Design Process

a. Design of view layer objects by applying Design Axioms and Corollaries.

b. Create prototype of the view layer interface.

3. Testing the usability and user satisfaction testing.

4. Iterate and refine the above steps.

11.19 USER INTERFACE DESIGN RULES

UI Design Rule 1:

Making the interface simple for complex application if the user interface is simple it is easy for the users to learn new applications. Each User Interface class should have a well define single purpose. If a user cannot sit before a screen and find out what to do next without asking multiple questions, then it says your interface is not simple.

UI Design Rule 2:

Making the Interface Transparent and Natural. The user interface should be natural that users can anticipate what to do next by applying previous knowledge of doing things without a computer. This rule says there should be a strong mapping and user's view of doing things.

UI Design Rule 3:

It allows users to be in control of the Software. The UI should make the users feel they are in control of the software and not the software controls the user. The user should play an active role and not a reactive role in the sense user should initiate the action and not the software.

Some ways to make put users in control are

1. Make the interface forgiving.
2. Make the interface visual.
3. Provide immediate feedback.
4. Avoid Modes.
5. Make the interface consistent.

11.20 PURPOSE OF VIEW LAYER INTERFACE

The user interface can employ one or more windows. Windows are used for the following purposes

- Forms and data entry windows
- Dialog boxes

11.21 GUIDELINES FOR DESIGNING FORMS AND DATA ENTRY WINDOW

- Identify the information which we want to display or change
- Identify the task that users need to work with data on the form or data entry window

Data entry tasks include

- Navigating rows in a table, such as moving forward and backward , and going to the first and last record
- Adding and deleting rows
- Changing data in rows
- Saving and cancelling the changes

Dialog Boxes

Dialog boxes display status information or ask users to supply information or make a decision before continuing with a task.

11.22 GUIDELINES FOR DESIGNING DIALOG BOXES AND ERROR MESSAGES

A dialog box provides an exchange of information or a dialog between the user and the application Dialog boxes generally appear after a particular Menu item or a Command button pressed Error message. If we wrongly enter the date in the entry form then the message show the format for date (DD/MM/YYYY)

11.23 GUIDELINES FOR THE COMMAND BUTTONS

Layout Position of the command buttons is very important

- Bottom
- Align top right
- Align left border is very popular in web interface

11.24 APPLICATION WINDOWS (MAIN WINDOW)

An application window is a container of application objects or icons. It contains an entire application with which users can interact.

Consist of Frame or border, Title bar, Scroll bars, Menu bar, Toll bar Status bar.

Check Your Progress

20. What is a dialog box?
21. Define application window.

11.25 ANSWERS TO CHECK YOUR PROGRESS QUESTIONS

1. A Database Management System is a set of programs that enables the creation and maintenance of a collection of related data.
2. The purpose of DBMS is to provide reliable, persistent data storage and mechanisms for efficient, convenient data access and retrieval.
3. A database model is a collection of logical constructs used to represent the data structure and data relationships within the database.
4. A primary key is a combination of one or more attributes whose value unambiguously (clearly) locates each row in the table.
5. DDL is the language used to describe the structure of and relationships between objects stored in a database.
6. A Data Manipulation Language (DML) is the language that allows users to access and manipulate (such as create, save or destroy) data organizations.
7. Logical database organization refers to the conceptual view of database structure and the relationships within the database.
8. Client programs manage user interface portion of the application, validate data entered by the user, dispatch requests to server program and executes business logic.
9. Application servers need not to be database centric. They may serve any of user needs such as sending mails, regulating download.

10. Cooperative processing is computing that requires two or more distinct processors to complete a single transaction. Here programs interact and execute concurrently on different processors
11. The CORBA common object environment (COE) provides system level services such as life cycle management for objects accessed through CORBA, event notification between objects and transaction and concurrency control.
12. The responsibility of an OODBMS includes definition of the object structures, object manipulation and recovery, which is the ability to maintain data integrity regardless of system, network or media failure.
13. Distributed databases imply that portions of the database reside on different nodes (computers) and disk drives in the network.
14. Distributed Object Computing (DOC) utilizes reusable software components that can roam anywhere on networks, run on different platforms, communicate with legacy applications by means of object wrappers.
15. A tuple (row) of a table contains data for a single entity that correlates to an object (instance of a class) in an object –oriented system.
16. Table-Class mapping is a simple one-to-one mapping of a table to a class and the mapping of columns in a table to properties in a class.
17. Heterogeneous information systems facilitate the integration of heterogeneous information sources, where they can be structured (having regular schema), semi-structured and sometimes even unstructured.
18. Open Data Base Connectivity (ODBC) is an application programming interface that provides solutions to the multi database programming problem.
19. The driver manager, loads and unloads drivers, performs status checks and manages multiple connections between applications and data sources.
20. Dialog boxes display status information or ask users to supply information or make a decision before continuing with a task.
21. An application window is a container of application objects or icons. It contains an entire application with which users can interact.

11.26 SUMMARY

- A package groups and manages the modeling elements, such as classes, their associations, and their structures.
- Packages themselves may be nested within other packages.
- A package may contain both other packages and ordinary model elements.

- Persistence refers to the ability of some objects to outlive the programs that created them.
- The persistent data are those data that exist beyond the lifetime of the creating process.
- Schema or meta-data contains a complete definition of the data formats, such as the data structures, types and constraints.
- The meta-data are usually encapsulated in the application programs themselves.
- A concurrency control policy dictates what happens when conflicts arise between transactions that attempt access to the same object and how these conflicts are to be resolved.
- A database model is a collection of logical constructs used to represent the data structure and data relationships within the database.
- The different database models are, Hierarchical model Network model Relational model.
- Open database connectivity (ODBC) is an application programming interface that provides solutions to the multidatabase programming problem.
- A multi database system (MDBS) is database systems that resides unobtrusively on top of existing relational and object databases, and file systems and presents a single database illusion to its users.
- Common object request broker architecture (CORBA) is a standard proposed as a means to integrate distributed heterogeneous business applications and data.
- Distributed object computing (DOC) utilizes reusable software components that can roam anywhere on networks, run on different platforms, communicate with legacy applications by means of object wrappers, and manage themselves and the resources.
- Cooperative processing is computing that requires two or more distinct processors to complete a single transaction.
- Data definition language (DDL) is the language used to describe the structure of and relationships between objects stored in a database
- View layer objects are more responsible for user interaction and these view layer objects have more relation with the user where business layer objects have less interaction with users.
- The user interface can employ one or more windows.
- An application window is a container of application objects or icons.

11.27 KEYWORDS

- Persistence

- Schema
- DDL
- DML
- ODBC
- CORBA
- Client Server Computing
- Distributed Object Computing

11.28 REVIEW QUESTIONS

1. Differentiate between transient data and persistent data?
2. What is a DBMS?
3. What is a relational database? Explain tuple, primary key and foreign key
4. What is a database schema? Differentiate between schema and meta-data
5. What is a DDL?
6. What is a distributed database?
7. What is concurrency control?
8. Define shareability
9. What is a transaction?
10. What is concurrency policy?
11. What is a query?
12. Define client-server computing
13. Name the different types of servers. Briefly describe.
14. Why is DOC so important in the computing world?
15. Describe CORBA, ORB and DCOM
16. What is an OODBMS? Differentiate between an OODBMS and object oriented programming.
17. Differentiate between forward and reverse engineering
18. Describe a federated multidatabase system
19. Describe the process of creating the access layer classes
20. Define object store and persistence.
21. Write a note on View layer.

11.29 FURTHER READINGS

1. Object Oriented Analysis and Design using UML, by Rational Software Corporation (2002) Bahrami, A.(1999).
2. Object Oriented Systems Development, using the unified modeling language, McGraw-Hill

UNIT XII – MANAGING ANALYSIS AND DESIGN

Structure

12.0 INTRODUCTION

To develop a strong system, we need a high level of confidence that

- Each component will behave correctly
- Collective behaviour is correct
- No incorrect behaviour will be produced

The elimination of the syntactical bug is the process of debugging while detection and elimination of the logical bug is the process of testing.

12.1 OBJECTIVES

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

- Understand about quality assurance tests
- Understand about testing strategies
- Describe the impact of testing

12.2 QUALITY ASSURANCE TESTS

Finding out which is wrong and correcting the code to eliminate the errors or bugs is called debugging process. Some of the kinds of errors are

Language errors: It is due to incorrectly constructed code. These are easiest type of errors. No need for debugging tools.

- Runtime errors: These are detected when the program is run.

- Logic errors: This occurs when the code produce incorrect results.

Quality assurance testing are divided into two categories

- Error based testing – Search a given class's method for particular clues of interest
- Scenario based testing – Also called usage-based testing. Concentrates on what the user does, not what the product does.

12.3 TESTING STRATEGIES

12.3.1 Black box testing

- An approach to testing where the program is considered as a 'black-box'
- The program test cases are based on the system specification
- Test planning can begin early in the software process
- In a black box, the test item is treated as "black" whose logic is unknown.
- All that's known is what goes in and what comes out, the input and output
- Black box test works very nicely in testing objects in an O-O environment.

12.3.2 White box testing

- White box testing assumes that specific logic is important, and must be tested to guarantee system's proper functioning.
- One form of white box testing is called path testing
- It makes certain that each path in a program is executed at least once during testing.

Two types of path testing are:

- Statement testing coverage – Tests every statement in the objects method by executing it at least once
- Branch testing coverage – Perform tests to confirm every branch alternative has been executed at least once.

12.3.3 Top Down testing

- It assumes that the main logic of the application needs more testing than supporting logic.
- Finds critical design errors in testing process and improves the quality of software
- Supports testing the user interface and event driven systems

12.3.4 Bottom up testing

- It assumes that individual programs and modules are fully developed as standalone processes.

- These modules are tested individually, and then combined for integration testing.

12.4 IMPACT OF OBJECT ORIENTATION ON TESTING

The impacts are as follows

- Some types of errors could become less plausible
- Some types of errors could become more plausible
- Some new types of errors might appear

12.4.1 Reusability of tests

Marick says that the simpler is a test, the more likely it is to be reusable in sub-classes. Simple tests find only faults. Complex tests find faults and also stumble across others.

What are the path testing's available in white box testing?

Check Your Progress

1. Define language errors.
2. What is scenario based testing?
3. What are the path testing's available in white box testing?

12.5 ANSWERS TO CHECK YOUR PROGRESS QUESTIONS

1. Language errors are due to incorrectly constructed code. These are easiest type of errors. No need for debugging tools.

2. Scenario based testing is also called as usage-based testing; it concentrates on what the user does, not what the product does.

3. Statement testing coverage and Branch testing coverage are the path testing's in white box testing.

12.6 SUMMARY

- The elimination of the syntactical bug is the process of debugging.
- Detection and elimination of the logical bug is the process of testing.
- Quality assurance testing is divided into two categories i) Error based testing ii) Scenario based testing
- White box testing assumes that specific logic is important, and must be tested to guarantee system's proper functioning.
- The concept of black box testing is used to represent a system whose inside workings are not available
- A top down strategy can detect the serious flaws early in the implementation.

12.7 KEYWORDS

- Testing
- Debugging
- Reusability

12.8 REVIEW QUESTIONS

1. Why quality assurance is needed?
2. What are the kinds of errors you might encounter when you run your program?
3. Define Black box testing
4. What is white box testing?
5. Explain top down testing.
6. What is Bottom-up Testing?
7. Summarize the impact of an object orientation on testing.
8. What are the kinds of errors you might encounter when you run your program?
9. Define Error based testing.

12.9 FURTHER READINGS

1. Object Oriented Analysis and Design using UML, by Rational Software Corporation (2002) Bahrami, A.(1999).
2. Object Oriented Systems Development, using the unified modeling language, McGraw-Hill.

NOTES

BLOCK 5 : UNIT XIII CODING AND MAINTENANCE

Structure

13.0 INTRODUCTION

Most developers are well aware of the concepts of object oriented development, which originates from entire software development life cycle. The Software development life cycle includes requirements, planning, design, coding, testing, deployment and maintenance and so on. Software maintenance is widely accepted part of SDLC now a days. It stands for all the modifications and updations done after the delivery of software product.

13.1 OBJECTIVES

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

- Understand about coding
- Understand about maintenance
- Describe object oriented metrics

13.2 CODING AND MAINTENANCE

- OOAD artifacts feed into implementation model in a traceable manner
- Some tools generate partial code from UML

- But programming not trivial generation!
- Programmers make changes as the work out the details
- Therefore, Expect and plan for change and deviation from design during programming
- Write source code for:
    - Class and interface definitions
    - Method definitions
- Work from OOA/D artifacts
    - Create class definitions for Domain Class Diagrams (DCDs)
    - Create methods from Interaction diagrams

13.2.1 Maintenance

There are four types of maintenance, namely, corrective, adaptive, perfective, and preventive.

Corrective maintenance is concerned with fixing errors that are observed when the software is in use.

Adaptive maintenance is concerned with the change in the software that takes place to make the software adaptable to new environment such as to run the software on a new operating system.

Perfective maintenance is concerned with the change in the software that occurs while adding new functionalities in the software.

Preventive maintenance involves implementing changes to prevent the occurrence of errors. The distribution of types of maintenance by type and by percentage of time consumed.

Corrective Maintenance

It deals with the repair of faults or defects found in day-today system functions. A defect can result due to errors in software design, logic and coding. Design errors occur when changes made to the software are incorrect, incomplete, wrongly communicated or the change request is misunderstood. Logical errors result from invalid tests and conclusions, incorrect implementation of design specifications, faulty logic or incomplete test of data. All these errors, referred to as residual errors. In the event of a system failure due to an error, actions are to be taken to restore the operation of the software system. Corrective maintenance approaches to locate the original specifications in order to determine what the system was originally designed to do. Corrective maintenance accounts for 20% of all the maintenance activities.

Adaptive Maintenance

It is the implementation of changes in a part of the system, which has been affected by a change that occurred in some other part of the system. Adaptive maintenance consists of adapting software to changes in the environment such as the hardware or the operating system. Adaptive maintenance accounts for 25% of all the maintenance activities.

NOTES

Perfective Maintenance

It deals with implementing new or changed user requirements. It involves making functional enhancements to the system in addition to the activities to increase the systems' performance even when the changes have not been suggested by faults. This includes enhancing both the function and efficiency of the code and changing the functionalities of the system as per the users changing needs. Perfective maintenance accounts for 50% of all the maintenance activities.

Preventive Maintenance

It involves performing activities to prevent the occurrence of errors. It tends to reduce the software complexity thereby improving program understandability and increasing software maintainability. It comprises documentation updating, code optimization and code restructuring. Documentation updating involves modifying the documents affected by the changes in order to correspond to the present state of the system. Code optimization involves modifying programs for faster execution or efficient use of storage space. Code restructuring involves transforming the program structure for reducing the complexity in source code and making it easier to understand. Preventive maintenance accounts for 5% of all the maintenance activities.

13.3 OBJECT-ORIENTED METRICS

Metrics can be broadly classified into three categories: project metrics, product metrics, and process metrics.

13.3.1 Project Metrics

Project Metrics enable a software project manager to assess the status and performance of an ongoing project. The following metrics are appropriate for object-oriented software projects

- Number of scenario scripts
- Number of key classes
- Number of support classes
- Number of subsystems

13.3.2 Product Metrics

Product metrics measure the characteristics of the software product that has been developed. The product metrics suitable for object-oriented systems are

Methods per Class − It determines the complexity of a class. If all the methods of a class are assumed to be equally complex, then a class with more methods is more complex and thus more susceptible to errors.

- **Inheritance Structure** − Systems with several small inheritance lattices are more well–structured than systems with a single large inheritance lattice. As a thumb rule, an inheritance tree should not have more than 7 (± 2) number of levels and the tree should be balanced.

- **Coupling and Cohesion** − Modules having low coupling and high cohesion are considered to be better designed, as they permit greater reusability and maintainability.
- **Response for a Class** − It measures the efficiency of the methods that are called by the instances of the class.

13.3.3 Process Metrics

Process metrics help in measuring how a process is performing. They are collected over all projects over long periods of time. They are used as indicators for long-term software process improvements. Some process metrics are

- Number of KLOC (Kilo Lines of Code)
- Defect removal efficiency
- Average number of failures detected during testing
- Number of latent defects per KLOC

Check Your Progress

1. List down the types of maintenance.
2. What is the purpose of adaptive maintenance?
3. Define code optimization.
4. Define code restructuring.
5. Define project metrics.
6. What is process metrics?

13.4 ANSWERS TO CHECK YOUR PROGRESS QUESTIONS

1. There are four types of maintenance namely, corrective, adaptive, perfective and preventive.
2. Adaptive maintenance is the implementation of changes in a part of the system, which has been affected by a change that occurred in some other part of the system.
3. Code optimization involves modifying programs for faster execution or efficient use of storage space.
4. Code restructuring involves transforming the program structure for reducing the complexity in source code and making it easier to understand.
5. Project Metrics enable a software project manager to assess the status and performance of an ongoing project.

NOTES

6. Process metrics help in measuring how a process is performing. They are collected over all projects over long periods of time.

13.5 SUMMARY

- There are four types of maintenance, namely, corrective, adaptive, perfective, and preventive.
- Corrective maintenance is concerned with fixing errors that are observed when the software is in use.
- Adaptive maintenance is concerned with the change in the software that takes place to make the software adaptable to new environment such as to run the software on a new operating system.
- Perfective maintenance is concerned with the change in the software that occurs while adding new functionalities in the software.
- Preventive maintenance involves implementing changes to prevent the occurrence of errors.
- Project Metrics enable a software project manager to assess the status and performance of an ongoing project.
- Product metrics measure the characteristics of the software product that has been developed.
- Process metrics help in measuring how a process is performing.

13.6 KEYWORDS

- DCD
- Maintenance
- Design errors
- Logical errors
- Project metrics

13.7 REVIEW QUESTIONS

1. What is the purpose of maintenance?
2. Define corrective maintenance.
3. What is adaptive maintenance?
4. Write a note on metrics.
5. Describe the various types of maintenance.

13.8 FURTHER READINGS

1. An overview of Object Oriented Design Metrics, Muktamyee Sarker, 2005
2. Object Oriented Analysis and Design using UML, by Rational Software Corporation (2002) Bahrami, A.(1999).

UNIT XIV – CASE STUDY

Structure

14.0 INTRODUCTION

The aim of a case study is to teach the student to observe the issue professionally and solve the problem of the definite case successfully. The cause and effect of the problem which has occurred and focus on the solution of the matter with the help of the reliable and valid methods. The success of the case study depends on the quality of the chosen solution.

14.1 OBJECTIVE

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

- Understand what a case study is
- Understand a particular topic clearly with cause and solution
- Understand the methods available to reach a solution

14.2 FOUNDATION CLASS LIBRARY

14.2.1. Library Management system

The case study is library management software for the purpose of monitoring and controlling the transactions in a library. This case study on the library management system gives us the complete information about the library and the daily transactions done in a Library. We need to maintain the record of news and retrieve the details of books available in the library which mainly focuses on basic operations in a library like adding new member, new books, and up new information, searching books and members and facility to borrow and return books. It features a familiar and well thought-out, an attractive user interface, combined with strong searching, insertion and reporting capabilities. The report generation facility of library system helps to get a good idea of which are borrowed by the members, makes users possible to generate hard copy.

The following are the brief description on the functions achieved through this case study:

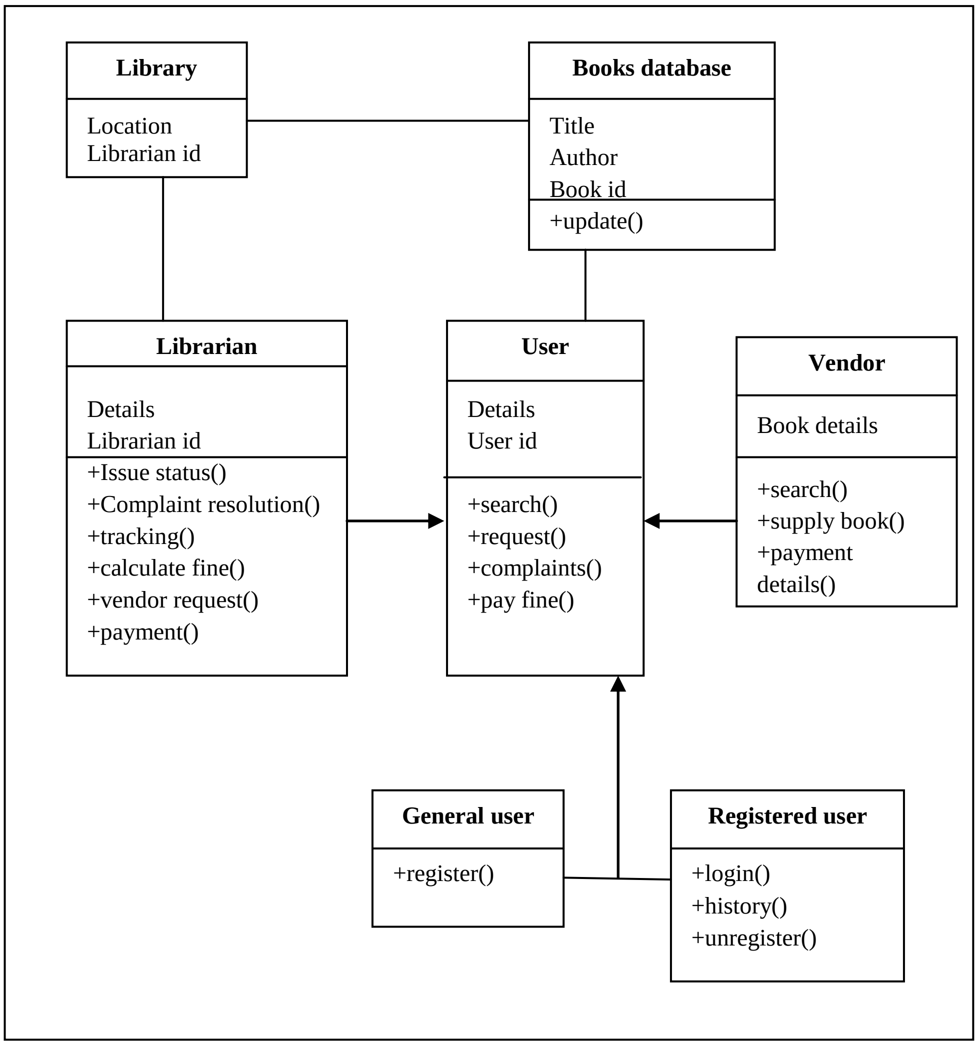
End-Users:

•Librarian: To maintain and update the records and also to cater the needs of the users.

•Reader: Need books to read and also places various requests to the librarian.

•Vendor: To provide and meet the requirement of the prescribed books.

Class Diagram



Actors vs Use Cases

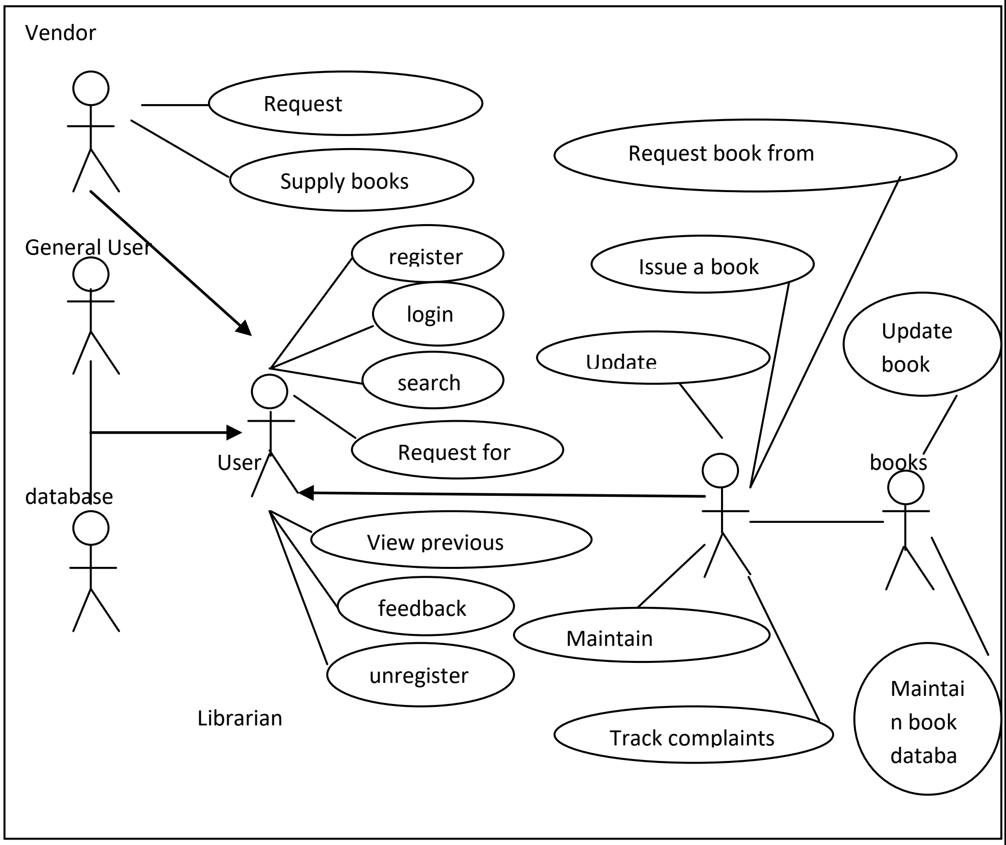
Librarian
•Issue a book
•Update and maintain records
•Request the vendor for a book
•Track complaints

User
•Register
•Login

•Search a book
•Request for isse
•View history
•Request to the Librarian
•Unregister

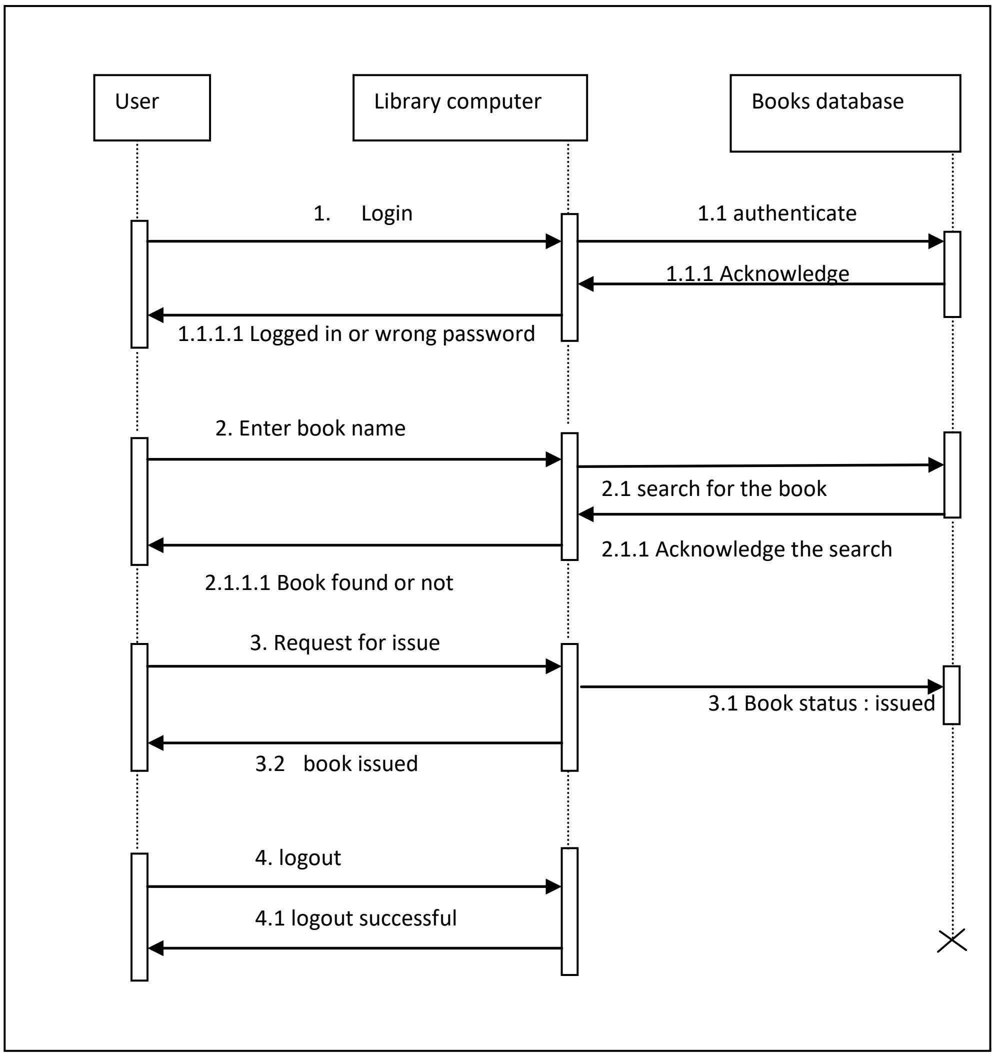
Books Database
•Update records
•Show books status

Vendors
•Provide books to the library
•Payment acknowledgement

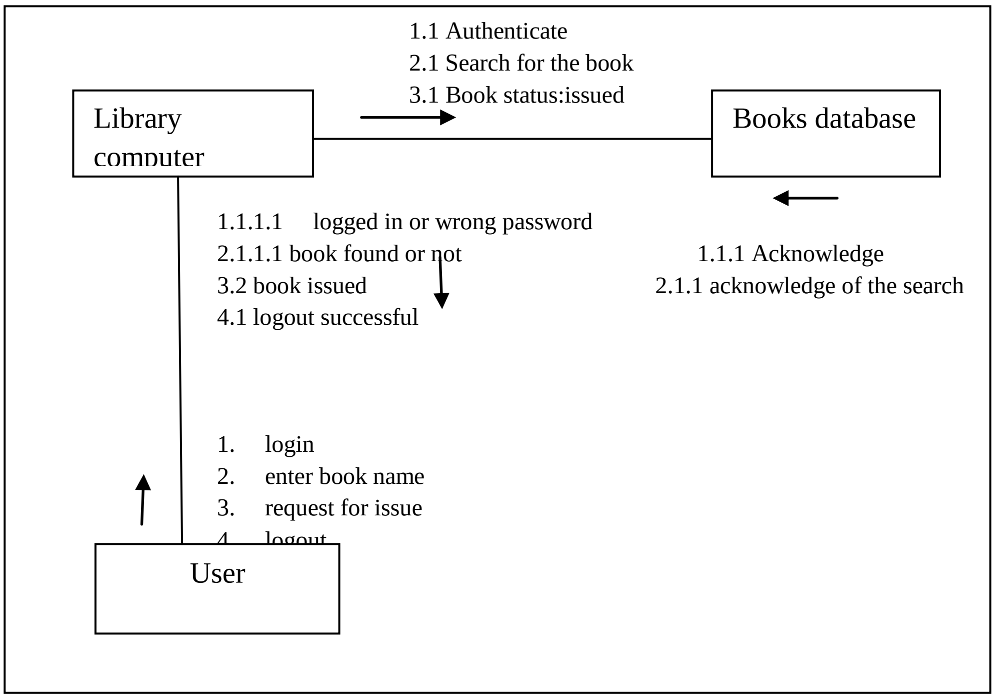


Sequence Diagram

Sequence diagram for searching a book and issuing it as per the request by the user from the librarian:

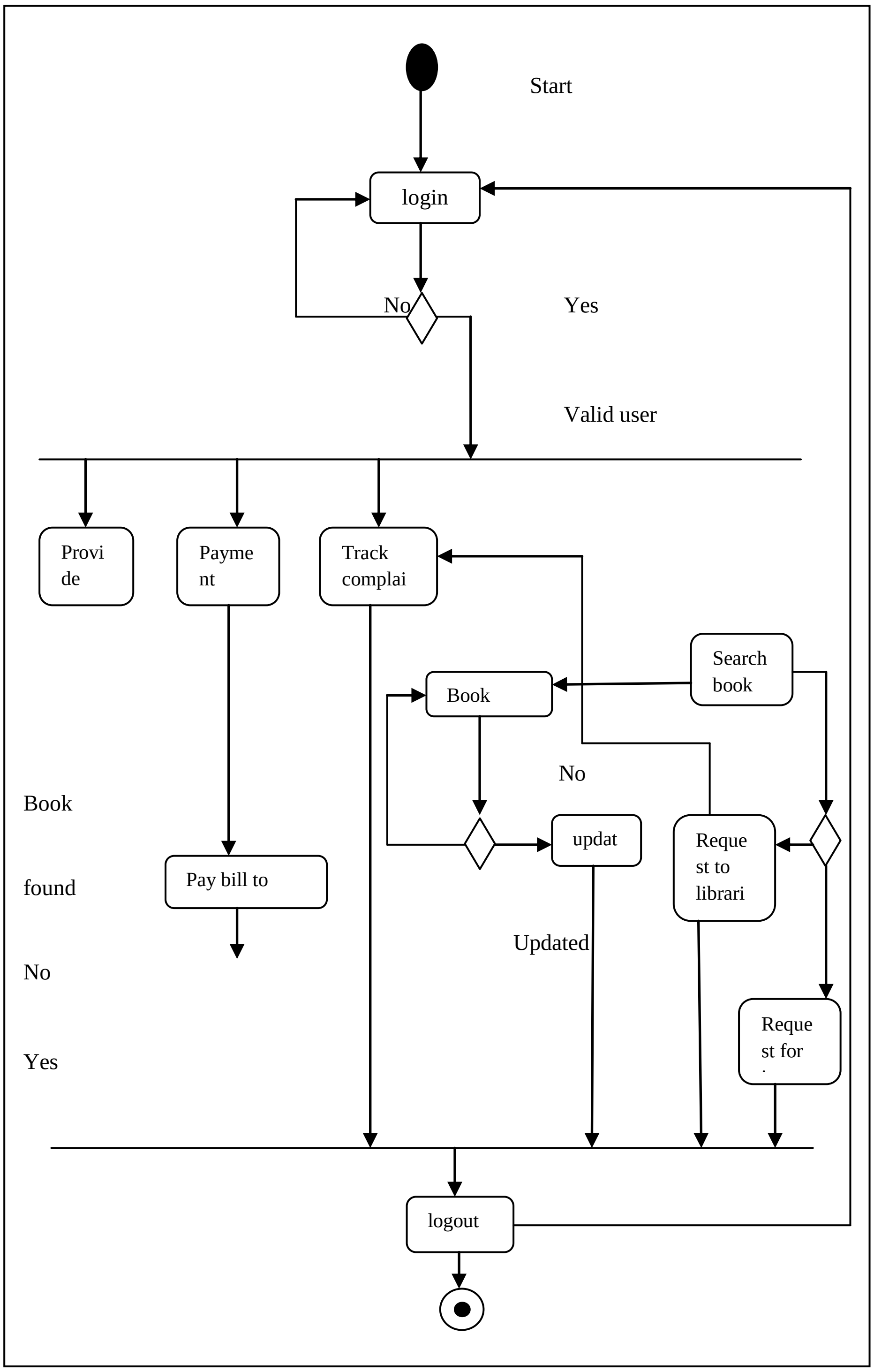


Collaboration Diagram for searching a book and issuing it as per the request by the user from the librarian:



Activity Diagram

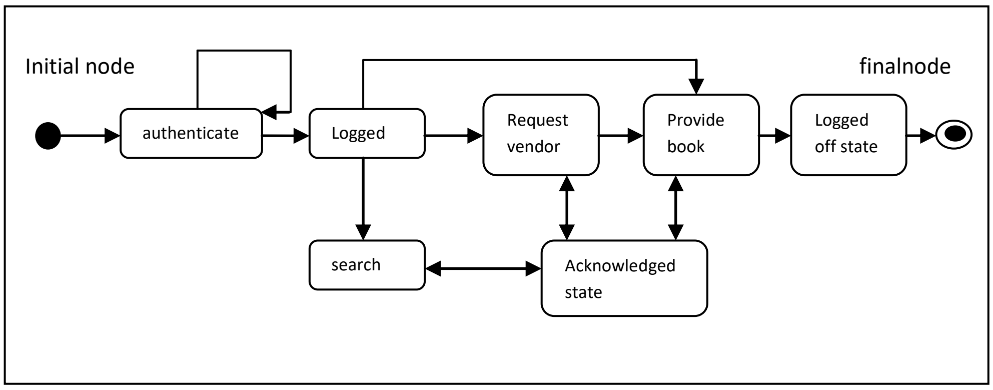
Activities:
User Login and Authentication
Search book operation for Reader
Acknowledge and Issue books to the users by the Librarian
Provide books requested by the Librarian from the Vendor
Bill payment from the Librarian to the Vendor
Status of the books updated in the Books Database



State Chart Diagram

States:
Authentication
Successfully logged on or re-login
Search for a book (user) / request the vendor (librarian) / provide the requested book (vendor)
Receive acknowledgement
Logged off / re-search / new function

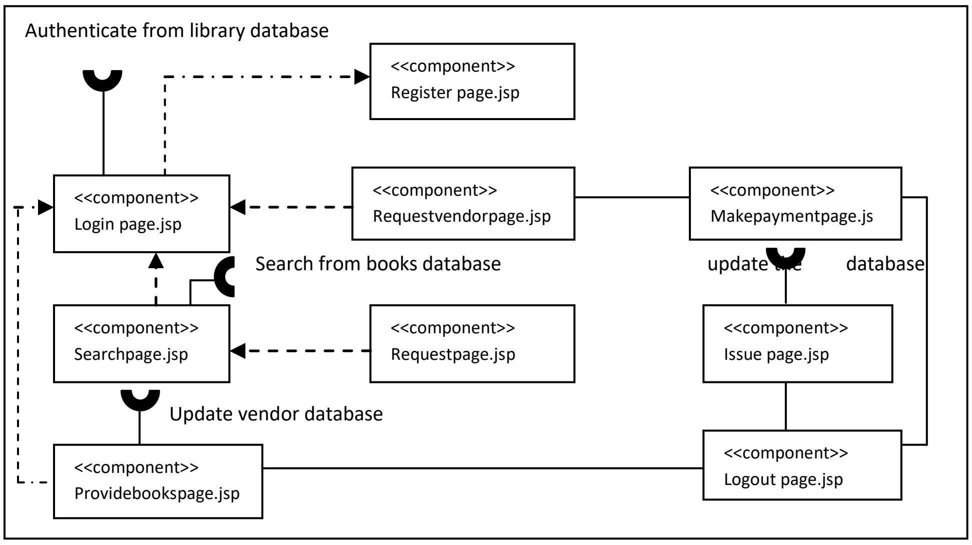
Transitions:
Authenticate ---> Logged in
Logged in ---> Search <---> Acknowledgement
Logged in ---> Request Vendor <---> Provide Book <---> Acknowledgement
Logged in ---> Provide Book <---> Acknowledgement
Acknowledgement ---> Logged off



Component Diagram

Components:

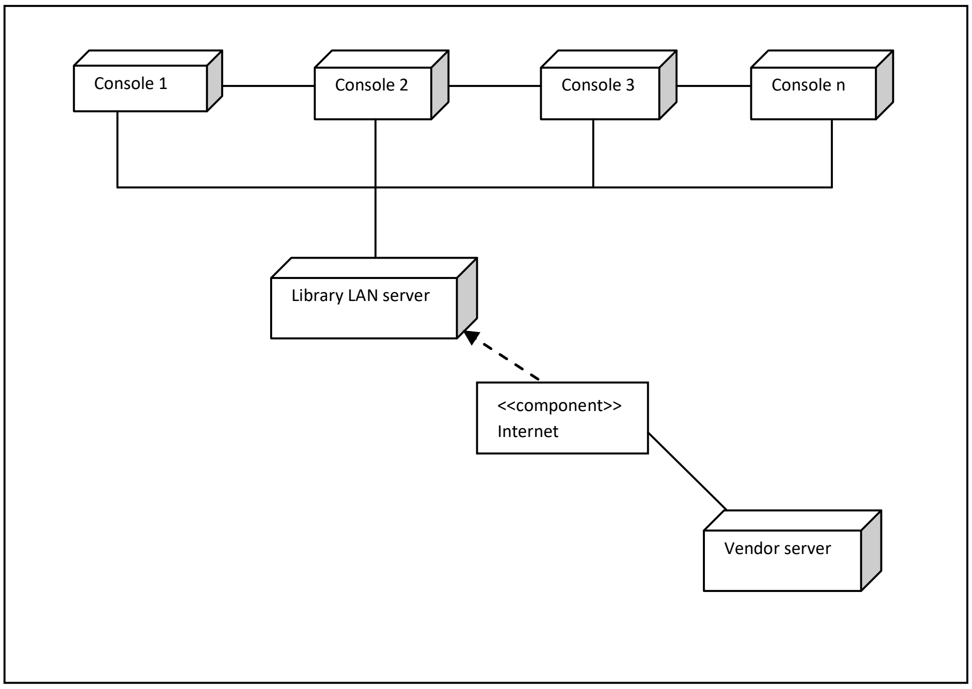
Register Page (visitor / vendor)
Login Page (user / librarian / vendor)
Search Page (user / librarian / vendor)
Request Vendor Page (librarian)
Request Book Issue Page (user / vendor)
Issue Status Page (librarian)
Make Payment Page (librarian / vendor)Provide Books Page (librarian)
Logout Page (user / librarian / vendor)


Authenticate from library database
<<component>> Register page.jsp
<<component>> Login page.jsp
<<component>> Requestvendorpage.jsp
<<component>> Makepaymentpage.js
Search from books database
update the database
<<component>> Searchpage.jsp
<<component>> Requestpage.jsp
<<component>> Issue page.jsp
Update vendor database
<<component>> Providebookspage.jsp
<<component>> Logout page.jsp

Deployment Diagram

Systems Used:

Local Consoles / Computers for login and search purposes by users, librarian and vendors.
Library LAN Server interconnecting all the systems to the Database.
Internet to provide access to Venodors to supply the requested books by the Librarian
Vendor Server to maintain the records of the requests made by the librarian and books provided to the library.



14.3 CLIENT / SERVER COMPUTING

Client server computing is the process of the creation of the special architectural patterns which maintain the functioning of the varied networks and connection of their components. The major components of client server computing are: the sets of servers which are the sources of information for everyone who applies for their help; the sets of clients, which take advantage of the services of servers; the networks which maintain the constant and quality connection between clients and servers and their interrelation. The client/server architecture is quite interesting, because clients and servers are not connected with anything; they function independently from one another. The client can use the services of different servers; they can use the one server but have no idea about the existence of the other one. The server is not bound with the client and can serve for a great number of clients simultaneously.

The client is able to apply for the services of various servers and possess no idea about the existence about other clients. The model of client server computing is defined according to the distribution of the duties of clients and servers. There are three levels of operations: the level of presentation of the information (the user's interface and the server's set of commands); the application level (the processing of the information); the level of data management (data storage and the access to the information). The history of the creation client/server architecture dates back to the 1960s and 1970s when the development of the Internet and computer networks found its birth. Since that time client server computing has been developing with constant alterations and improvement. Client server computing is the important activity which enables the regular functioning of the Internet and the student is able to analyze the problem in the appropriate way.

DISTANCE EDUCATION
MCA DEGREE EXAMINATION
OBJECT ORIENTED ANALYSIS AND DESIGN
MODEL QUESTION PAPER

Time: 3 Hours Maximum Marks: 75

PART – A (10 x 2 = 20)

ANSWER ALL QUESTIONS

1. What is the purpose of Object orientation?
2. Define State.
3. Define Framework.
4. Give the difference between user and actor.
5. What is association?
6. How do you identify attributes?
7. State different types of coupling.
8. Expand: CORBA
9. Give the advantages of access layer.
10. What is the impact of testing?

PART – B (5 x 5 = 25)

ANSWER ALL QUESTIONS

11. a. Write a note on Object basics.

Or

b. Explain the relationship among classes.

12. a. Elucidate Booch methodology.

Or

b. Explain Use case model.

13. a. Explicate Noun phrase approach.

Or

b. Describe Super- Sub class relationships.

14. a. Explain Database model.

Or

b. Write a note on Distributed databases & Client server computing.

15. a. Elucidate testing strategies.

Or

b. Write a note on types of Maintenance.

PART- C (3 x 10 = 30)

ANSWER ANY THREE QUESTIONS

16. Describe the elements of object model.
17. Explicate software development life cycle.
18. Write a note on UML class diagram.
19. Write a brief note on Corollaries.
20. Explain in detail Object relation mapping.