CS F213
Object Oriented Programming

BITS Pilani
Hyderabad Campus

Prof.R.Gururaj
CS&IS Dept.

Object Oriented Analysis and Design (OOAD)

OO SW Development

It is a different way of thinking about SW based on abstraction that exists in the real world.

If we adopt OO approach for Analysis, Design and Development then the methodology is called as Object Oriented Development approach.

OO Analysis

Models problem domain using Object-Oriented concepts, leading to understanding of the problem. Objects represent entities in the problem domain (called semantic objects).

OO Design

Models solution to the problem in terms of objects.

The goal is to identify the objects that the system contains, relationships and interactions between the objects.

(Identify Semantic + other objects).

OO Implementation

Includes programming, testing and deployment of the SW.

OO Analysis

The goal of this phase is to give a complete description of WHAT the SW product should do.

The result of analysis phase is a textual description called *Requirements Specifications* (*Functional and Non-functional)*

Functional specs define the task to be performed,

readable both by the domain expert and SW developer.

Diverse interested parties can review this document.

It can be tested against reality.

Functional Requirements: Describe what to be done or what is the functionality of the system.

Functional requirements include:

- Process the system carries out.
- Input
- Output
- Details about the data

Non-Functional requirements Covers the aspects that describe how well the system provides the functionality.

- Performance
- Throughput
- Security

Design

Design is concerned with establishing how to deliver the functionality that was specified in Functional Specs, while at the same time, meeting non-functional requirements that may sometimes conflict with each other.

OO Design

The goal of this phase is to identify

- Classes
- Responsibilities of these classes (responsibilities are not methods, but high level functionality)
- Relationships among these classes

Classes and Objects

innovate achieve lead

Class : defines the structure and behavior of similar kind of objects.

Object: is an entity in a program that belong to a particular class

Each object has the following characteristics:

State (information stored by the object)

Behaviour (is defined by operations supported by the object that change the state of the object)

Identity (to distinguish between similar class objects)

Steps

- *Problem description*
- Identifying *classes*
- Identifying *responsibilities*
- Identify *Relationships* between classes
- Identify *Use Cases*
- *Identify Collaborations*
- Define *interactions*

Problem description for telephone voice-mail system

In a voice mail system, a person dials an extension number and, provided the other party does not pick up the telephone, leaves a message. The other party can later retrieve the messages, keep them, delete them. Real-world systems have a multitude of fancy features: messages can be forwarded to one or more mail boxes; distribution lists can be defined, retained and edited; and authorized persons can send broadcast messages to all users.

Identifying Classes

Look at the problem description

Thumb rule: look for *nouns* in the problem description.

Nouns:

Mailbox

Message

User

Passcode

Extension

Class names should be nouns in singular form

Identifying *responsibilities*

Look at the problem description

Thumb rule: look for *verbs* in the problem description.

Messages are recorded, deleted, played

Users log-in

Passcodes are checked

Add Message to Mailbox is the responsibility of Mailbox

We must find one class that owns the responsibility.

A responsibility must belong to exactly one class.

Relationships between classes

Three common relationships are :

Dependency, Aggregation, Inheritance

Dependency (coupling)

A class depends on another if it manipulates objects of another class, or call methods of other class objects.

It is easy to understand or implement if a class has no dependency.

Ex: Mailbox manipulates Message objects.

As far as possible-minimize coupling.

Aggregation- A class aggregates another if it contains objects of another class.

Ex: MessageQueue aggregates Message objects.

Inheritance- A class inherits from another if it incorporates the behavior of the other class.

Ex: ForwardedMessage inherits Message class.

Forwarded message contains specific info. Who forwarded it.

Use cases

Use cases are an analysis technique to describe in a formal way how a system should work.

Each Use case focuses on describing the steps that are necessary to achieve a behavior and bring it to successful completion.

Each step represents an interaction with people or entities outside the system (actors) and the system itself.

Use cases list sequence of steps that yields a result that is of value to an actor

Ex:

Leave a message - describes steps that caller must take to dial an extension and leave the message. Caller is the actor here.

Retrieve messages – the mail box owner is the actor.

A use case should include *variants* that describe some failures in achieving a result.

All such variants are known as *use case scenarios*.

Use case- *leave a message*

innovate achieve lead

Steps:

1. Caller dials the main number of voice mail system
2. VMS speaks a prompt (Enter mailbox number followed by#)
3. The Caller types in the extension#
4. VMS speaks (you have reached the mailbox of xxxx, please leave a message)
5. User speaks the message
6. The caller hangs-up
7. VMS places the recorded message in the recipient's mailbox.

CRC Cards

- A CRC card is an index card that describes a class, its high-level responsibilities, and its collaborations (dependent classes).
- It is a design technique for discovering class, responsibility and relationships.

Index cards are good because:

1. They are small (discourage piling up too much responsibility into one class)
2. Low-tech
3. Convenient to handle during brain storming sessions

Guidelines

1. Don’t assign more than three responsibilities (high-level) to a class
2. Maintain coherency (attach only related responsibilities)
3. Don’t add unnecessary responsibilities
4. Delete classes with no responsibilities

Notes:

- CRC cards are suited for group discussions.
- Is a mechanism for discovering classes
- Not a means of documenting design

Collaborations

Defines how objects collaborate to provide a functionality.

Hence one class may participate in more than one collaboration.

Ex. CRC Cards

Mailbox	
Keep new and saved messages	**MessageQueue**

MailSystem	
Manage mailboxes	**Mailbox**

Interactions

OO programs work by sending/receiving messages from/to objects.

Message is a request by sender to execute an operation of a target object.

This kind of message passing is called *as* *interaction* between objects.

Benefits of OOAD

The object oriented approach is a way of thinking about a problem using real world concepts instead using ad-hoc function concepts.

We intend to learn OOAD approach for the following reason:

- Promotes better understanding of user requirements
- Leads cleaner design
- Design flexibility
- Decomposition
- Facilitates data abstraction & information hiding
- Software reuse
- Easy maintenance

UML
(Unified Modeling Language)

What is a Model?

A model is a simplification of reality

UML

- Graphical notation for conveying design info.

- Three well known researchers- Booch (Booch) Jacobson (OOSE) and Rumbaugh (OMT)and, got together to unify their OO design notations and developed UML, *the unified modeling language*

Work on UML began in late 1994, at Rational Software Corporation.

Standardized by OMG in 1997. UML 1.0

- Popular method for designing OO SW.
- It is for visualizing, specifying, constructing, and documenting the artifacts of OO Software systems.

UML Diagrams

innovate achieve lead

1. Use Case Diagram
2. Class Diagram
3. Sequence Diagram
4. Collaboration Diagram
5. State Diagram
6. Activity Diagram
7. Object Diagram
8. Component Diagram
9. Deployment Diagram

Views in UML:

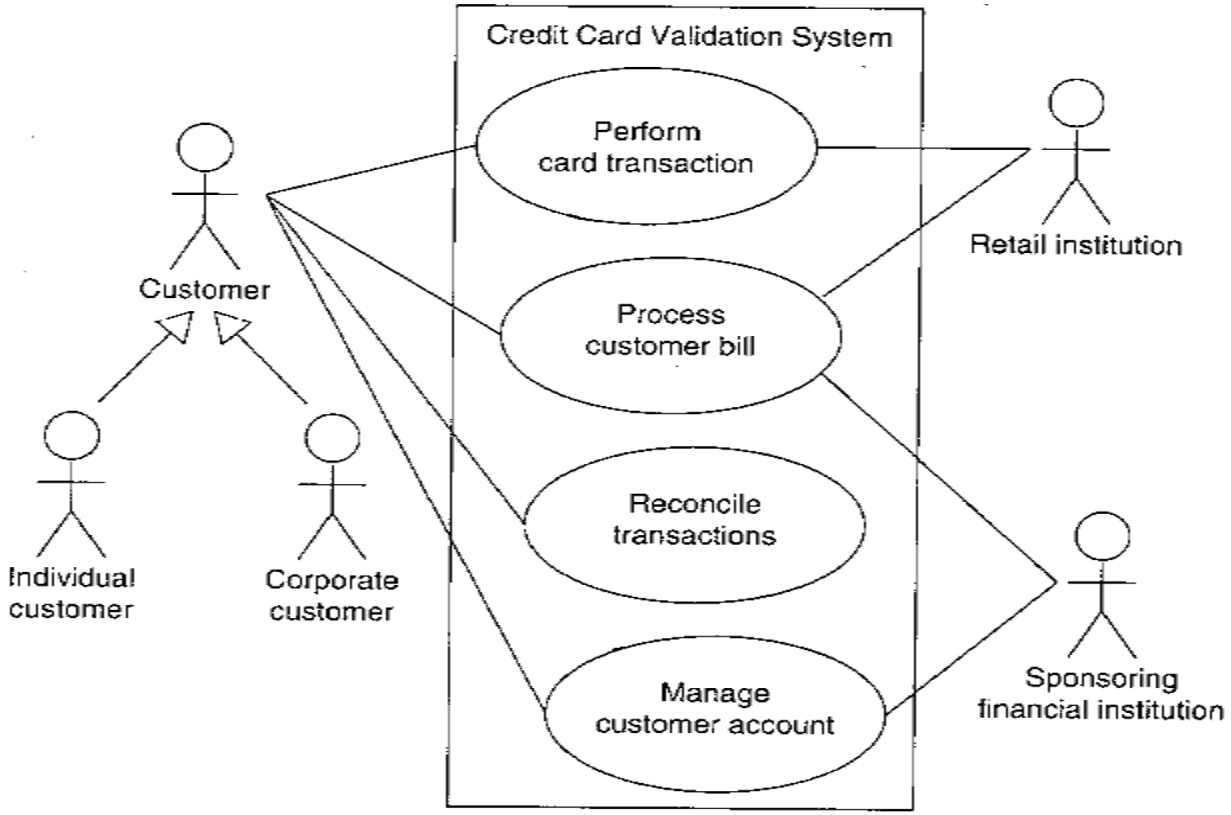
- Use case view:
  - Use case diagrams.
- Static or Structural view:
  - Class diagrams.
- Behavioral or Dynamic view:

  Sequence, Collaboration, State and Activity diagrams.
- Organizational or Implementation view:

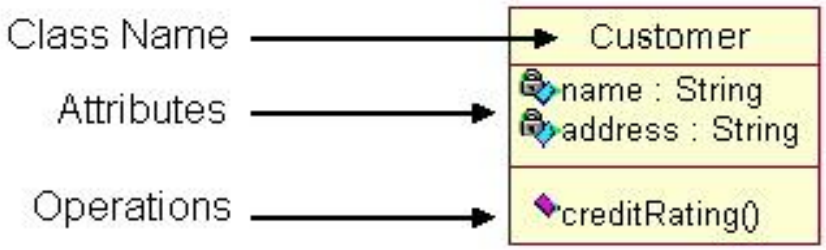
  Component diagrams, Node diagrams.
- Deployment view:
  - Deployment diagrams.

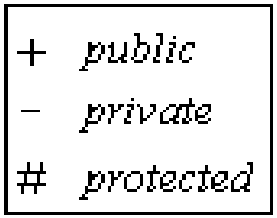
Use case Diagram

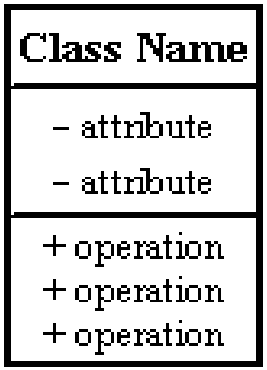
Use cases are an analysis technique to describe in a formal way how a system should work.



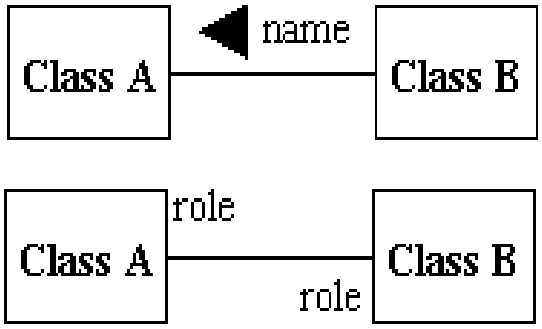
Class Diagram



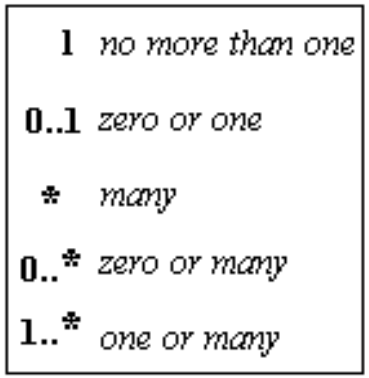




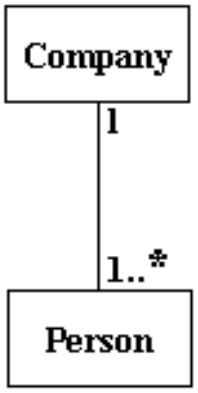
Relationships:

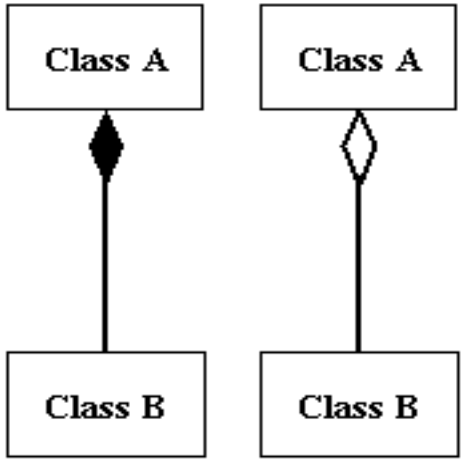


Association



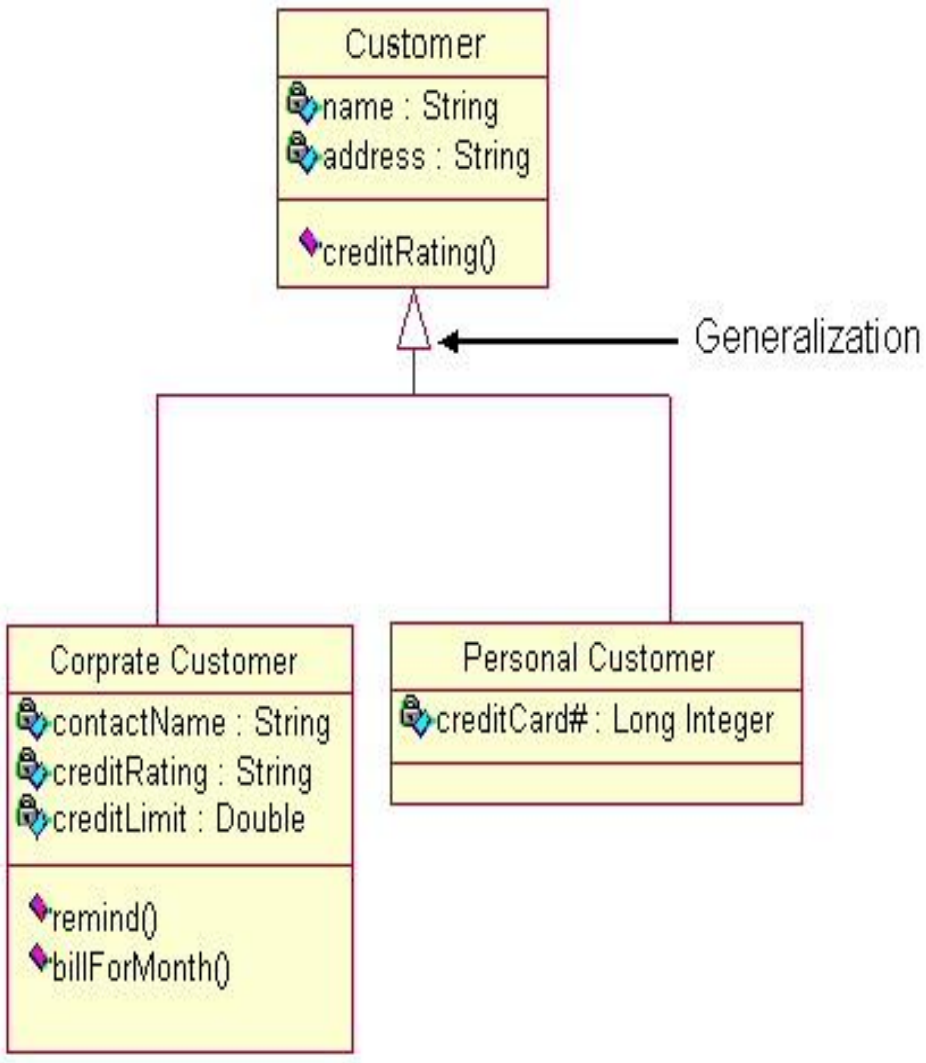
Multiplicity





Composition Aggregation

Generalization:



Object name : Class

Named object

: Class

Unnamed object

Object name : Class::Package

Named object with path name

Object diagrams

emp1 : Employee

Named objects

: Employee

Unnamed objects

emp1 : Employee::myproject

Named objects with path

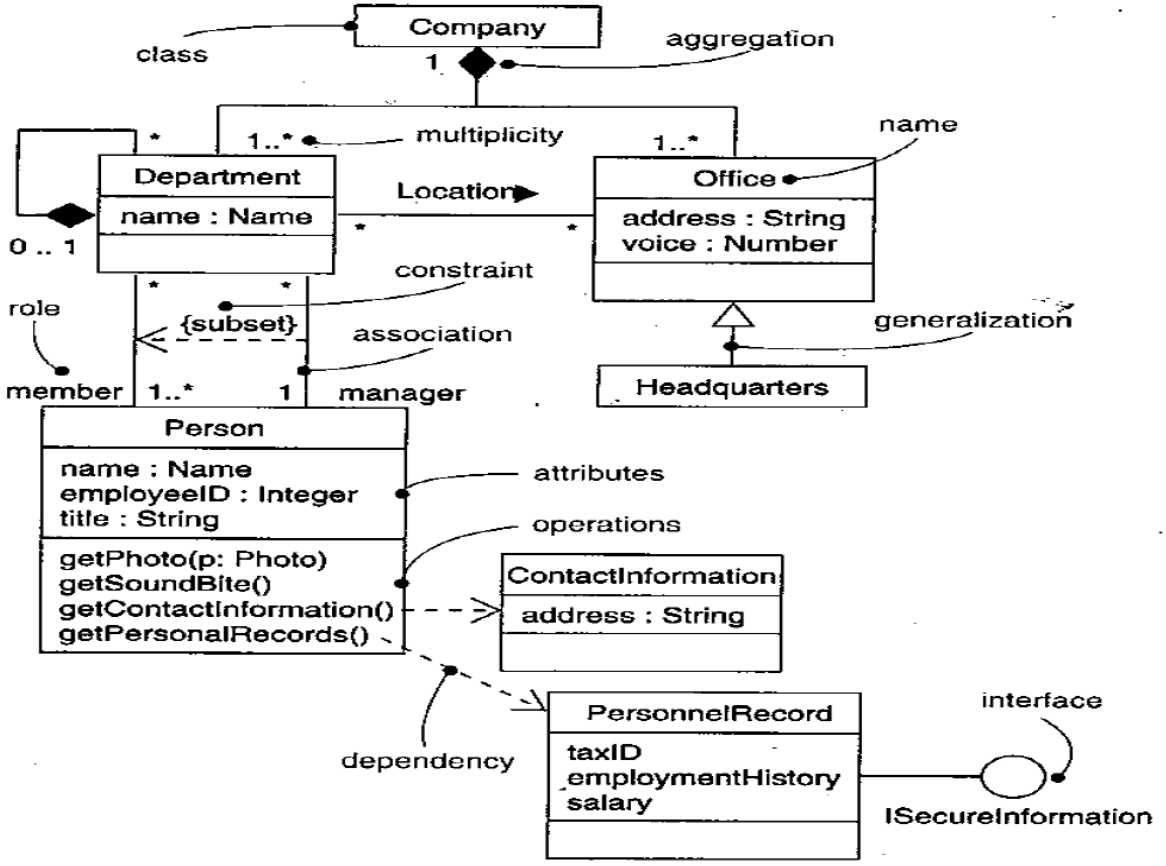


Figure 8-1: A Class Diagram

Object diagrams:

Show set of objects and their relationships.

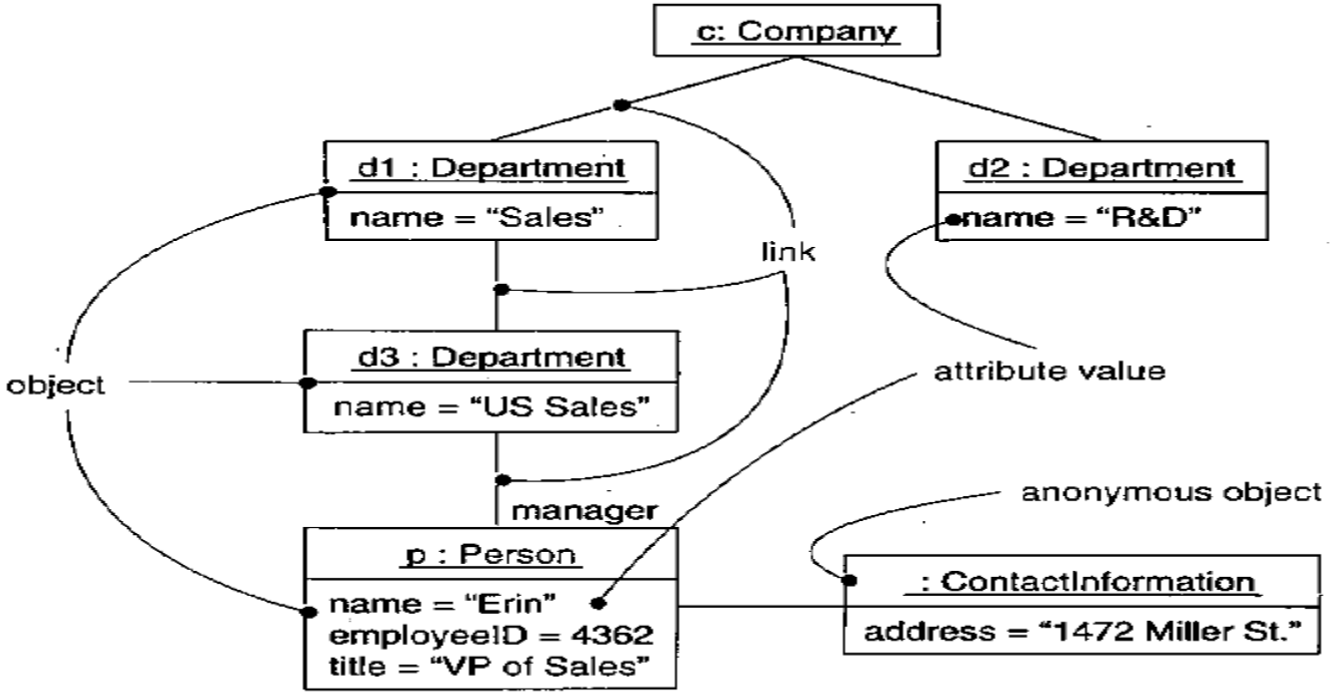


Figure 14-1: An Object Diagram

Sequence diagram:

- A sequence diagram shows the time ordering of sequence of method calls.
- Shows communication pattern among objects.
- Represent message sequencing in the system (time ordering of messages)

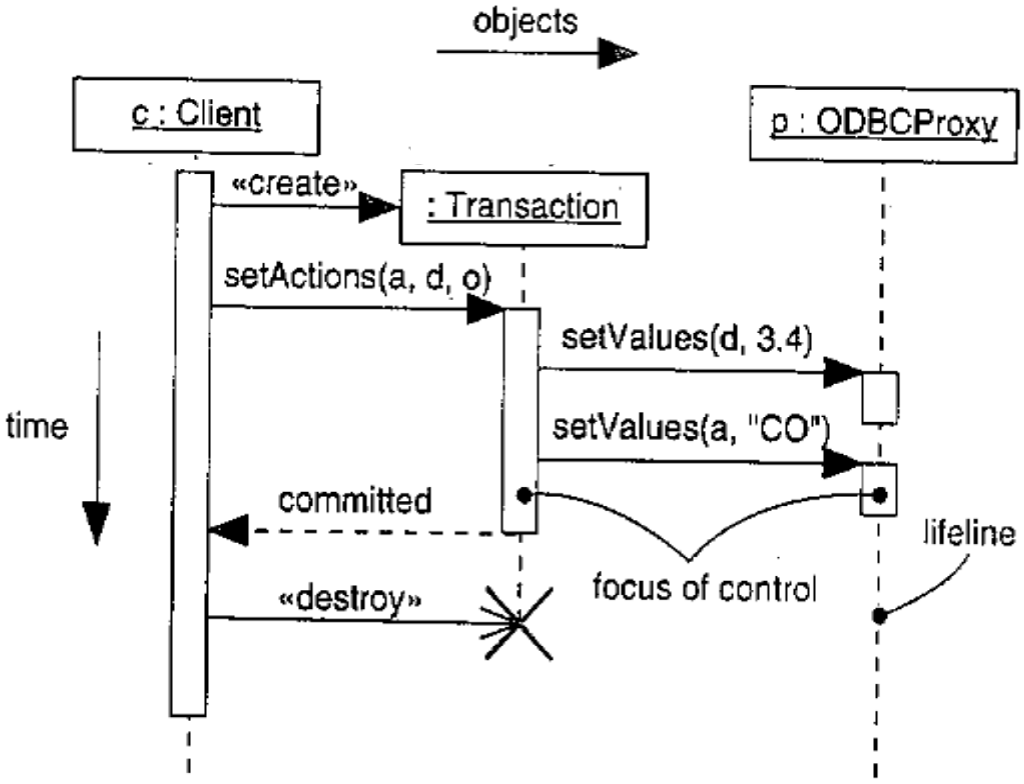


Figure 18-2: Sequence Diagram

Collaboration diagram:

- Similar to sequence diagram (interaction).
- Emphasizes the structural organization of objects that participate in the interaction.

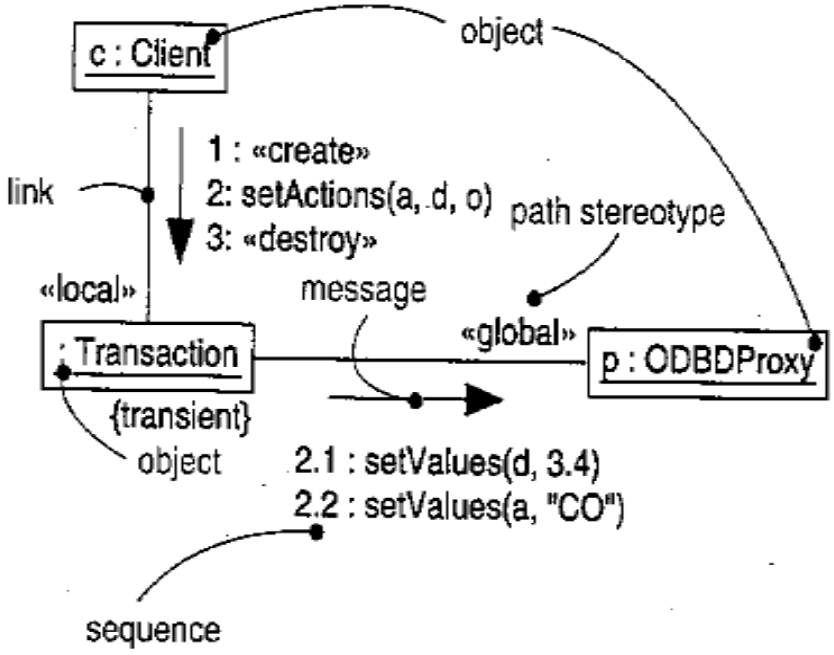


Figure 18-3: Collaboration Diagram

Collaboration diagrams have two features that distinguish them from sequ

Statechart Diagram:

innovate achieve lead

A state diagram shows the states of an object and the transitions between states.

State diagrams model the possible life history of an object.

They describe the behavior of a system .

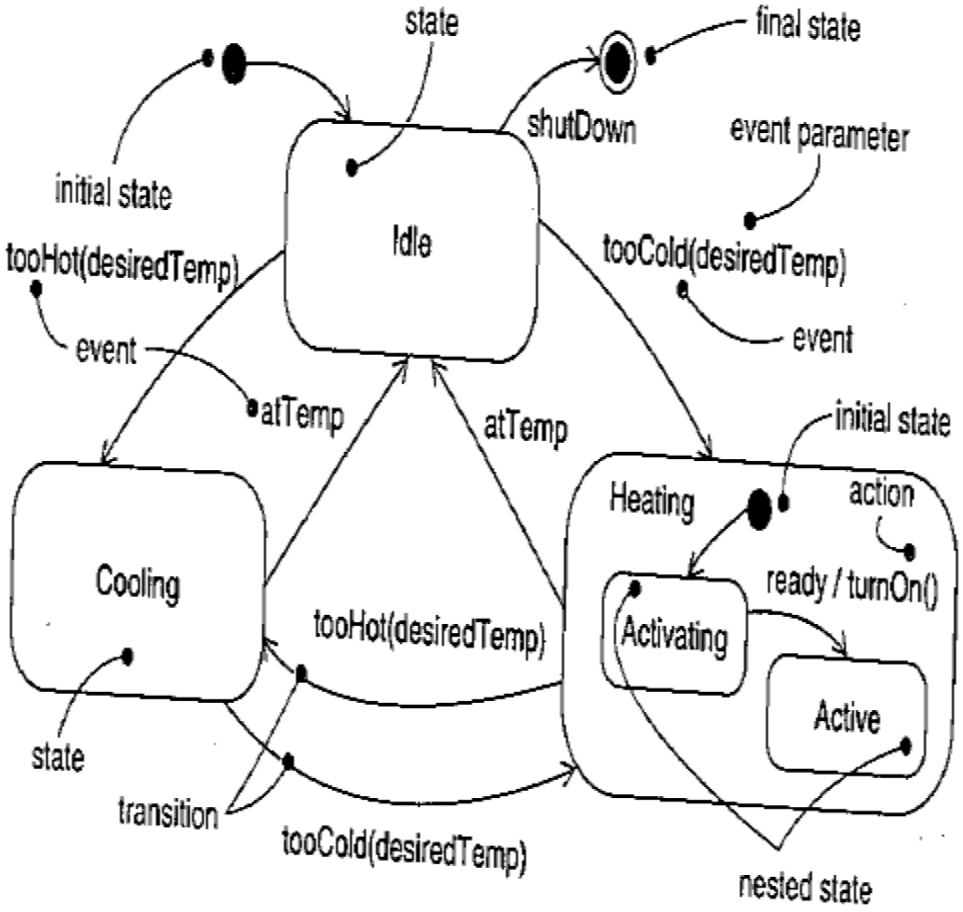


Figure 21-1: State Machines

Activity diagram:

Is a special kind of statechart diagram that shows flow from activity within a system.

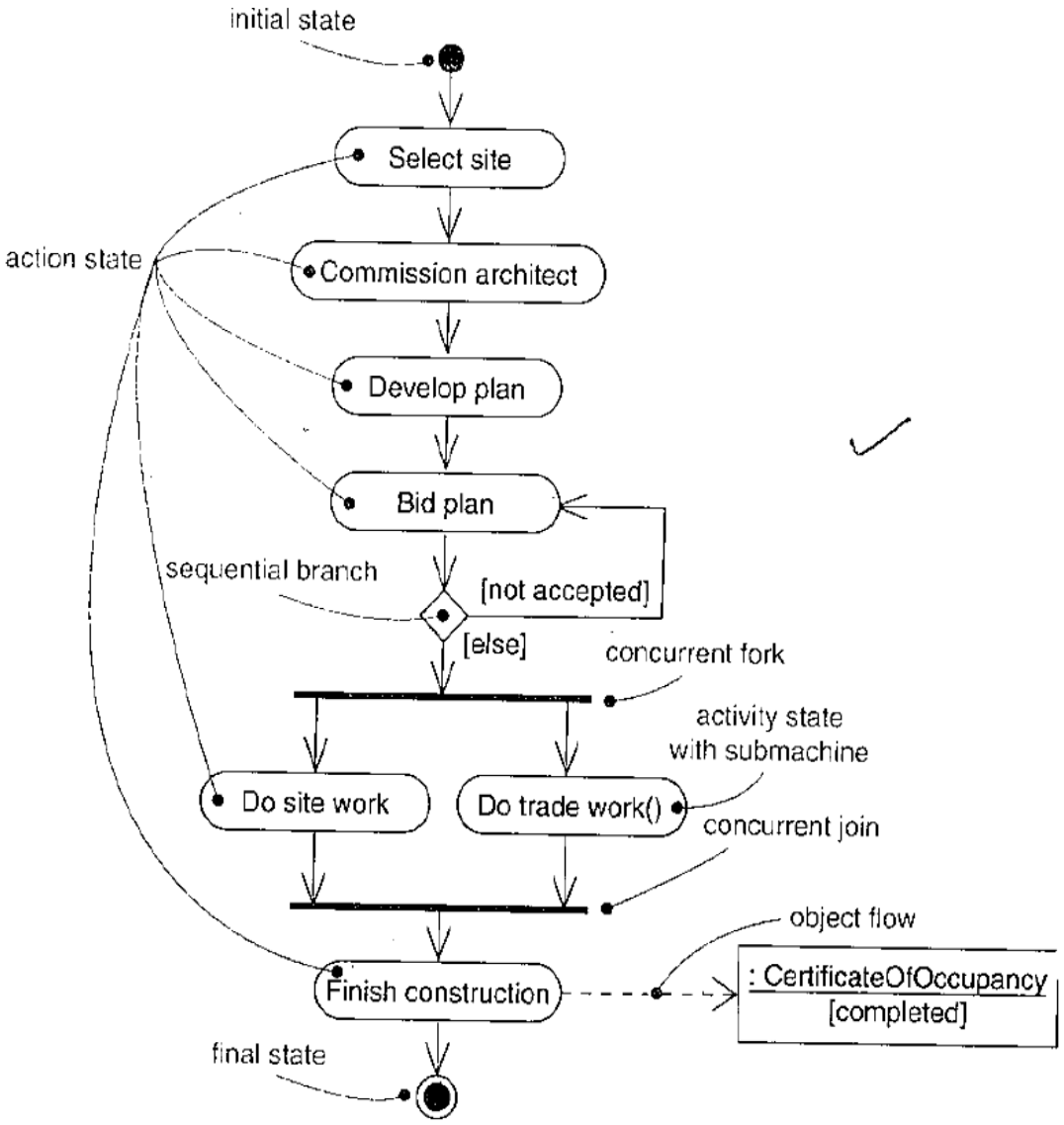


Figure 19-1: Activity Diagrams

Component diagram:

Organization and dependencies among the components.

Component is a physical and replaceable part of the system that conforms to and provides the realization of set of interfaces.

Packages

Libraries

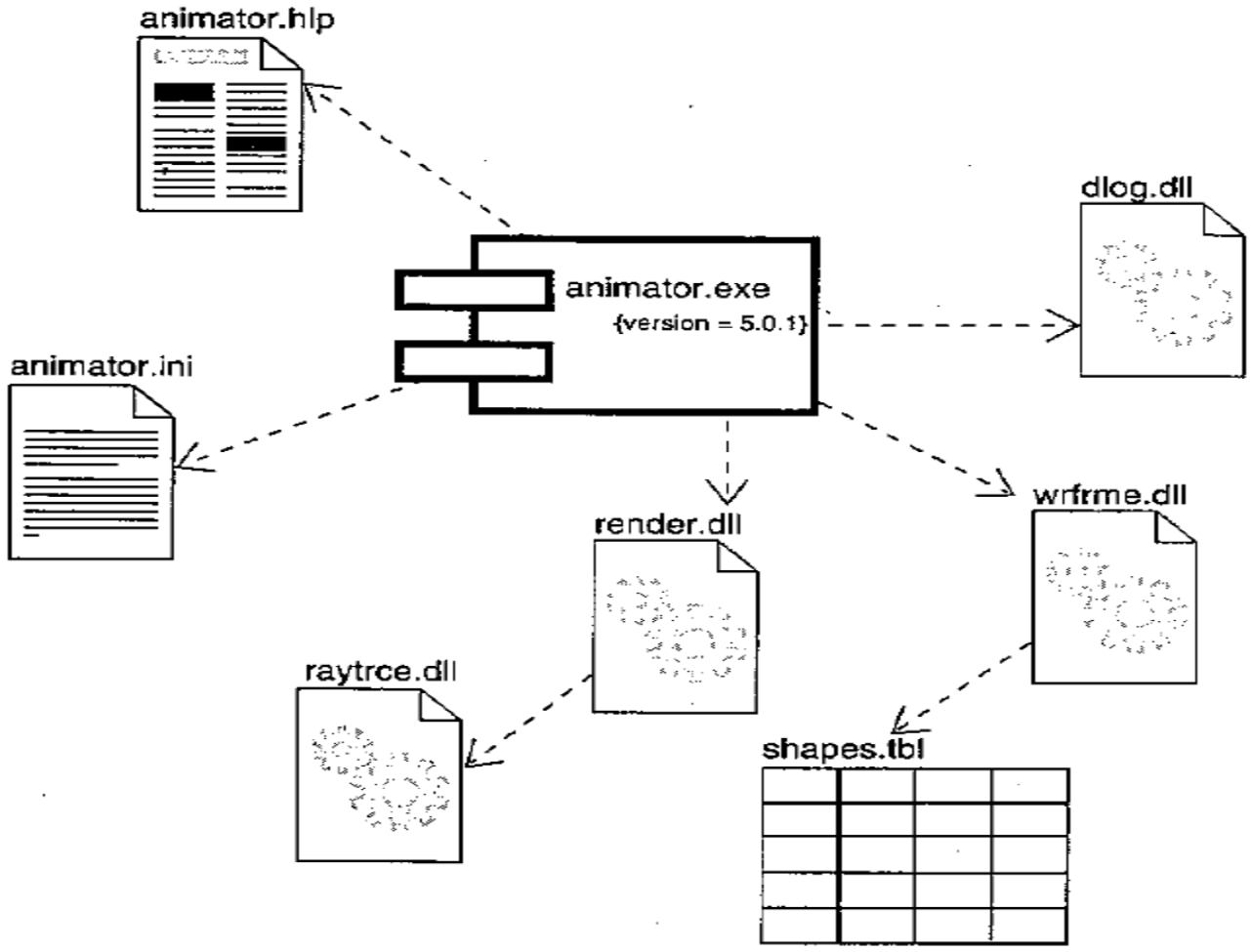
Class files

File

Database tabale

Executable file

Etc


animator.hlp
animator.exe
{version = 5.0.1}
dlog.dll
animator.ini
render.dll
wrfrme.dll
raytrce.dll
shapes.tbl

Deployment diagram:

Shows configuration of run-time processing nodes and the components that live in them. Address the static deployment view

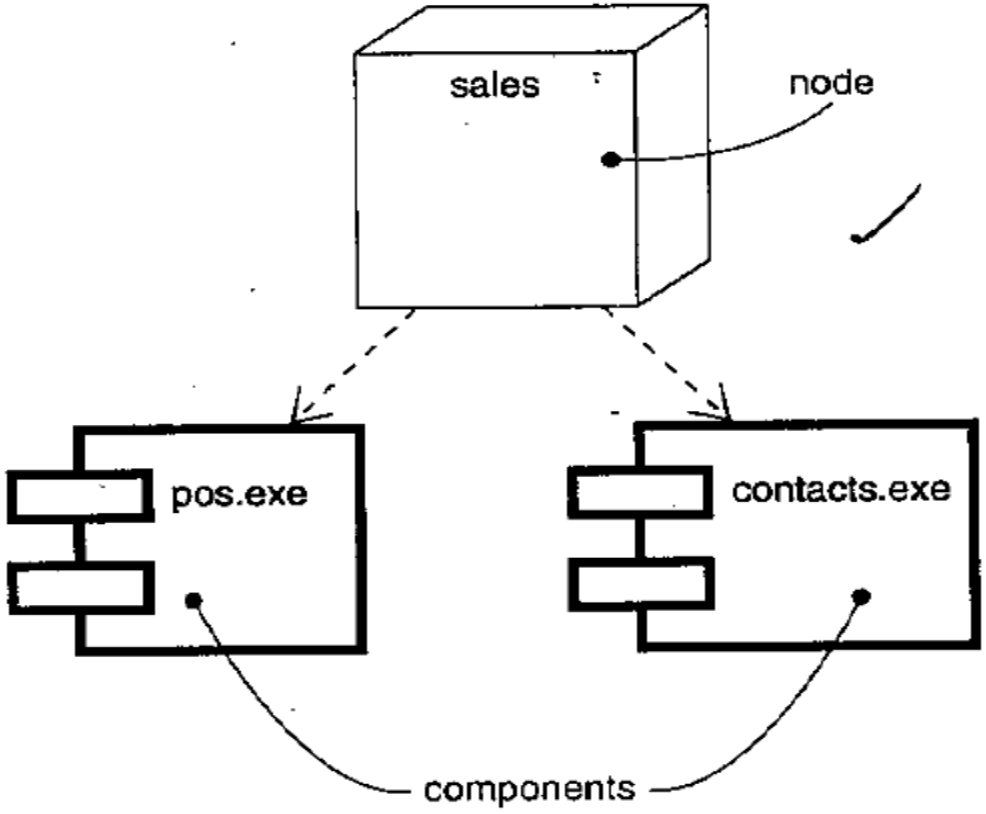
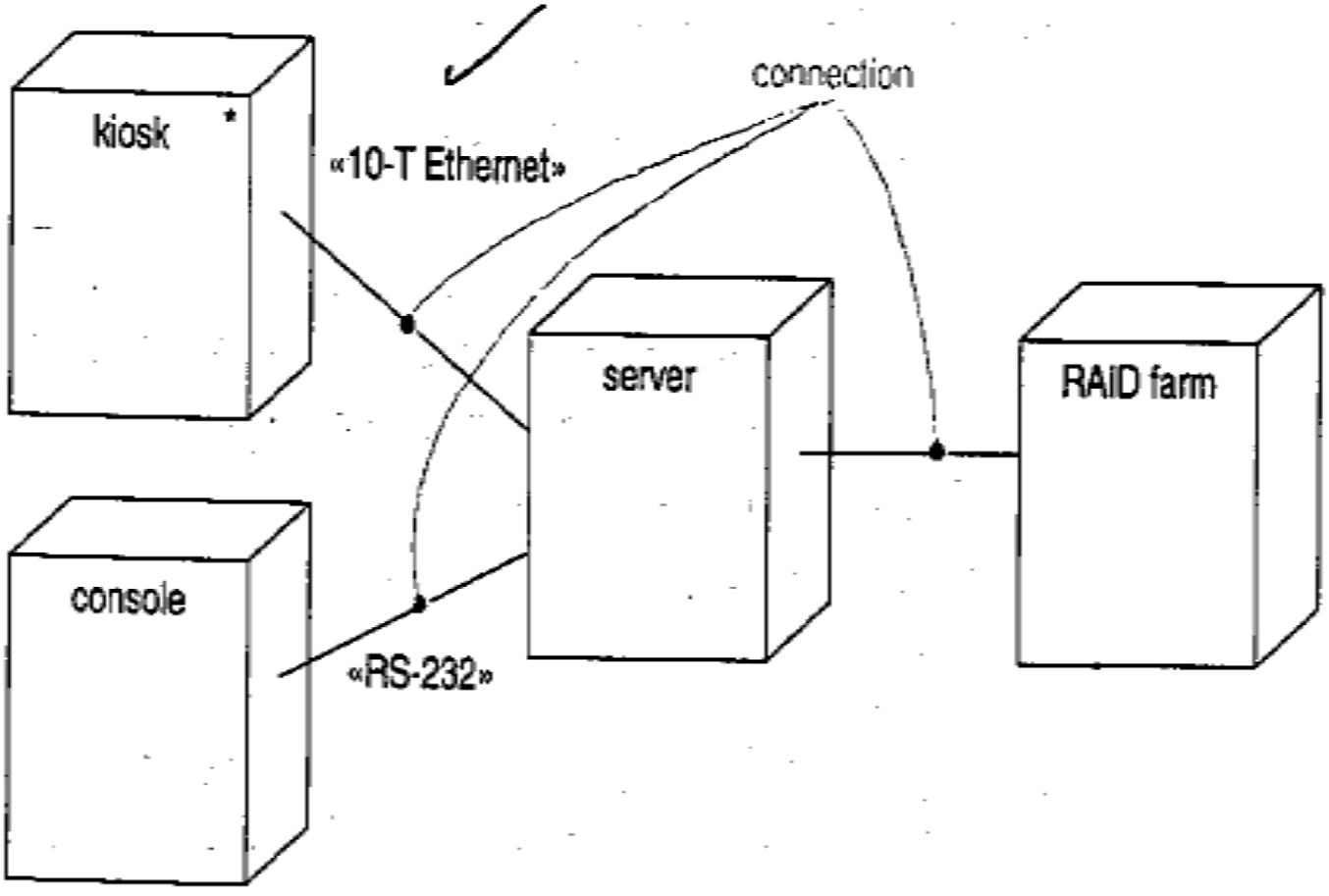


Figure 26-3: Nodes and Components


kiosk
«10-T Ethernet»
connection
server
RAID farm
console
«RS-232»

Rational Rose (IBM):

A UML visual modeling and application development solution (tool).

The IBM® **Rational Rose**® product family is designed for Unified Modeling Language (UML) based development of applications.

We can also use the following free UML Tools

- Visual Paradigm
- StarUML
- UMLet
- Violet UML Editor

Exercise

Assume a Library Management System (LMS) where users use the same for different purposes.

Users: Admin user- can login and perform adding a users, delete users, generate stock report.

A normal user can login, search for book, take issue, return, pay penalty.

User has userID, login, name, address, contact.

Book has book ID, book title, author.

Admin has userID, login, Designation etc.

Normal user has category,Start date.

Summary

- OOA
- OOD
- Design phases
- Identifying classes, responsibilities, use cases.
- CRC cards
- Interaction
- UML
- UML Diagrams and Notations.

References

R1. Object Oriented Analysis and Design with Applications, Grady Booch, Addison Wesley,

2nd Edition.

R2. The Unified Modeling Language User Guide, the ultimate tutorial to the UML from the Original

Designers, G Booch, J Rumbaugh, I Jacobson, Pearson Education, 2006.

And Class notes.