Computer Science & Engineering Department

B.Tech 2nd Year 4th Semester

session(2018-2019)

Software Engineering Lab (RCS-452)

S.No	List of the Practicals
1	Prepare a SRS document in line with the IEEE recommended standards.
2	Draw the Entity relationship diagram of a project.
3	Draw the data flow diagrams at level 0 and level 1.
4	Draw use case diagram in argo UML.
5	Draw activity diagram in argo UML.
6	Draw class diagram in argo UML.
7	Draw the component diagram in argo UML.
8	Draw sequence diagram in argo UML.
9	Draw collaboration diagram in argo uml.
	Value Added Practical:
1	Use testing tool such as junit.
2	Using configuration management tool-libra.

Ms. Rohit Aggarwal & Mrs.Deepika Gupta
(Signature of Faculty Lab Incharge)

Prof. Pradeep Pant
(HOD-CSE)

Software Engineering LAB (RCS-452)

PO1: An ability to use the methodology and modern engineering tools necessary for engineering practice.

PO2: ability to elicit, analyze and specify software requirements.

PO3: Analyze and translate specifications into a design.

Experiment Name: **Software Requirement Specification.**

Outcome: Can produce the requirements in a SRS document**.**

Objective: Prepare a SRS document in line with the IEEE recommended standards.

Description:

An SRS is basically an organization's understanding (in writing) of a customer or potential client's system requirements and dependencies *at a particular point in time* (usually) prior to any actual design or development work. It's a two-way insurance policy that assures that both the client and the organization understand the other's requirements from that perspective at a given point in time.

The SRS document itself states in precise and explicit language those functions and capabilities a software system (i.e., a software application, an eCommerce Web site, and so on) must provide, as well as states any required constraints by which the system must abide. The SRS also functions as a blueprint for completing a project with as little cost growth as possible. The SRS is often referred to as the "parent" document because all subsequent project management documents, such as design specifications, statements of work, software architecture specifications, testing and validation plans, and documentation plans, are related to it.

It's important to note that an SRS contains functional and nonfunctional requirements only; it doesn't offer design suggestions, possible solutions to technology or business issues, or any other information other than what the development team understands the customer's system requirements to be.

A well-designed, well-written SRS accomplishes four major goals:

> It provides feedback to the customer. An SRS is the customer's assurance that the development organization understands the issues or problems to be solved and the software behavior necessary to address those problems. Therefore, the SRS should be written in natural language (versus a formal language, explained later in this article), in an unambiguous manner that may also include charts, tables, data flow diagrams, decision tables, and so on.

It decomposes the problem into component parts. The simple act of writing down software requirements in a well-designed format organizes information, places borders around the problem, solidifies ideas, and helps break down the problem into its component parts in an orderly fashion.

It serves as an input to the design specification. As mentioned previously, the SRS serves as the parent document to subsequent documents, such as the software design specification and statement of work. Therefore, the SRS must contain sufficient detail in the functional system requirements so that a design solution can be devised.

It serves as a product validation check. The SRS also serves as the parent document for testing and validation strategies that will be applied to the requirements for verification.

SRSs are typically developed during the first stages of "Requirements Development," which is the initial product development phase in which information is gathered about what requirements are needed--and not. This information-gathering stage can include onsite visits, questionnaires, surveys, interviews, and perhaps a return-on-investment (ROI) analysis or needs analysis of the customer or client's current business environment. The actual specification, then, is written after the requirements have been gathered and analyzed.

SRS should address the following

The basic issues that the SRS shall address are the following:

- ***Functionality***. What is the software supposed to do?
- ***External interfaces***. How does the software interact with people, the system's hardware, other hardware, and other software?
- ***Performance***. What is the speed, availability, response time, recovery time of various software functions, etc.?
- ***Attributes***. What are the portability, correctness, maintainability, security, etc. considerations?
- ***Design constraints imposed on an implementation***. Are there any required standards in effect, implementation language, policies for database integrity, resource limits, operating environment(s) etc.

Characterstics of good SRS

An SRS should be

a) Correct
b) Unambiguous
c) Complete
d) Consistent
e) Ranked for importance and/or stability
f) Verifiable
g) Modifiable
h) Traceable

- **Correct** - This is like motherhood and apple pie. Of course you want the specification to be correct. No one writes a specification that they know is incorrect. We like to say - "Correct and Ever Correcting." The discipline is keeping the specification up to date when you find things that are not correct.
- **Unambiguous -** An SRS is unambiguous if, and only if, every requirement stated therein has only one interpretation. Again, easier said than done. Spending time on this area prior to releasing the SRS can be a waste of time. But as you find ambiguities - fix them.
- **Complete -** A simple judge of this is that is should be all that is needed by the software designers to create the software.
- **Consistent -** The SRS should be consistent within itself and consistent to its reference documents. If you call an input "Start and Stop" in one place, don't call it "Start/Stop" in another.
- **Ranked for Importance -** Very often a new system has requirements that are really marketing wish lists. Some may not be achievable. It is useful provide this information in the SRS.
- **Verifiable -** Don't put in requirements like - "It should provide the user a fast response." Another of my favorites is - "The system should never crash." Instead, provide a quantitative requirement like: "Every key stroke should provide a user response within 100 milliseconds."
- **Modifiable -** Having the same requirement in more than one place may not be wrong - but tends to make the document not maintainable.
- **Traceable -** Often, this is not important in a non-politicized environment. However, in most organizations, it is sometimes useful to connect the requirements in the SRS to a higher level document. Why do we need this requirement?

A sample of basic SRS

1. **Introduction**
1.1 Purpose

1.2 Document conventions

1.3 Intended audience

1.4 Additional information

1.5 Contact information/SRS team members

1.6 References

2. **Overall Description**
2.1 Product perspective
2.2 Product functions
2.3 User classes and characteristics
2.4 Operating environment
2.5 User environment
2.6 Design/implementation constraints
2.7 Assumptions and dependencies

3. **External Interface Requirements**
3.1 User interfaces

3.2 Hardware interfaces

3.3 Software interfaces

3.4 Communication protocols and interfaces

4. **System Features**
4.1 System feature

4.1.1 Description and priority
4.1.2 Action/result

4.1.3 Functional requirements 4.2 System feature B

5. **Other Nonfunctional Requirements**
5.1 Performance requirements

5.2 Safety requirements

5.3 Security requirements

5.4 Software quality attributes

5.5 Project documentation

5.6 User documentation

6. **Other Requirements**
Appendix A: Terminology/Glossary/Definitions list

Appendix B: To be determined

Output: SRS FOR ATM

SOFTWARE REQUIREMENT SPECIFICATION

ATM

Version 1.0

September 8, 2006

AN AUTOMATED TELLER MACHINE

Table of Contents

1 Introduction

The software **ATMExcl 3.0™** version1.0 is to be developed for Automated Teller Machines (ATM). An automated teller machine (ATM) is computerized telecommunications device that provides a financial institution's customers a secure method of performing financial transactions, in a public space without the need for a human bank teller. Through **ATMExcl 3.0™** ,customers interact with a user-friendly interface that enables them to access their bank accounts and perform various transactions.

1.1 Purpose

This SRS defines External Interface, Performance and Software System Attributes requirements of **ATMExcl 3.0™**. This document is intended for the following group of people:-

Developers for the purpose of maintenance and new releases of the software.
Management of the bank.
Documentation writers.
Testers.

1.2 Scope

This document applies to Automated Teller Machine software **ATM 3.0™**. This software facilitates the user to perform various transaction in his account without going to bank. This software offers benefits such cash withdrawals, balance transfers, deposits, inquiries, credit card advances and other banking related operations for customers. It also allows the administrator to fix the tariffs and rules as and when required.

The software takes as input the login Id and tha bank account number of the user for login purposes. The outputs then comprise of an interactive display that lets the user select the desirable function that he wants to perform..

The software is expected to complete in duration of six months and the estimated cost is Rs18 lakhs.

1.3 Definitions, Acronyms, and Abbreviations.

AC	Alternate Current
AIMS	ATM Information Management System.
ATM	An unattended electronic machine in a public place, connected to a data system and related equipment and activated by a

	bank customer to obtain cash withdrawals and other banking services.
Braille	A system of writing and printing for blind or visually impaired people, in which varied arrangements of raised dots representing letters and numerals are identified by touch.
BMS	Bank Management Software developed by KPM Bank.
CDMA	Code Division Multiple Access, a reliable data communication protocol.
CMS	Card Management Software developed by KPM Bank.
DES	Data Encryption Standard.
Dial-Up POS	A message format for low cost communications.
Electronic Journals	For easier, safer information storage, related to modem.
Internet	An interconnected system of networks that connects computers around the world via the TCP/IP protocol.
MB	Mega Bytes
ms	Milliseconds.
sec	Seconds
Smart Card	Card without hardware which stores the user's private keys within a tamper proof software guard.
SRS	Software Requirements Specification.

Tactile keyboard	Special keyboard designed to aid the visually impaired.
TCP/IP	Transmission Control Protocol/Internet Protocol.
V	Volts
VGA	Video Graphics Adaptor is a display standard.

1.4 References

The references for the above software are as follows:-
www.google.co.in
www.winkipedia.com
IEEE. Software Requirements Specification Std. 830-1993.
Chevy Chase Bank, UMBC Branch.
Russell C. Bjork Requirements Statement for Example ATM System. Online.
URL: http://www.math-cs.gordon.edu/local/courses/cs211/ATMExample/

1.5 Overview

Section 1.0 discusses the purpose and scope of the software.
Section 2.0 describes the overall functionalities and constraints of

the software and user characteristics.

Section 3.0 details all the requirements needed to design the software.

2. The Overall Description

2.1 *Product Perspective*

The ATM is a single functional unit consisting of various sub-components.

This software allows the user to access their bank accounts remotely through an ATM without any aid of human bank teller.

This software also allows the perform various other functions apart from just accessing his bank account such as mobile bill clearings etc.

Some of its hardware components are cassettes, memory, drives, dispensers i.e. for receipts and cash, a card reader, printer, switches, a console, a telephone dialer port, a networking port and disks.

The ATM communicates with the bank's central server through a dial-up communication link.
The Memory of the system shall be 20MB.

The Cassette capacity shall be at least 2000 notes.

2.2 *Product Functions*

The major functions that **ATMExcel 3.0™** performs are described as follows:-

Language Selection:- After the user has logged in, the display provides him with a list of languages from which he can select any one in order to interact with the machine throughout that session. After the language selection the user is prompted with an option that whether he wants the selected language to be fixed for future use so that he is not offered with the language selection

menu in future thus making the transaction a bit faster. User also has the freedom to switch to a different language mentioned in the list in between that session.

Account Maintenance:- The various functions that a user can perform with his account are as follows:-

Account Type:-The user has the freedom to select his account type to which all the transactions are made.

Number: 2

Experiment Name: Entity Relationship Diagram

Hardware Requirements: Pentium 4 processor (2.4 GHz), 128 Mb RAM, Standard keyboard n mouse, colored monitor.

Software Requirements: SmartDraw, MS Word.

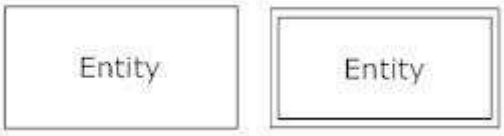
Outcome: Can produce the requirements in Entity Relationship diagram**.**

Objective: Prepare a Requirement document in by using Entity Relationship Diagram.

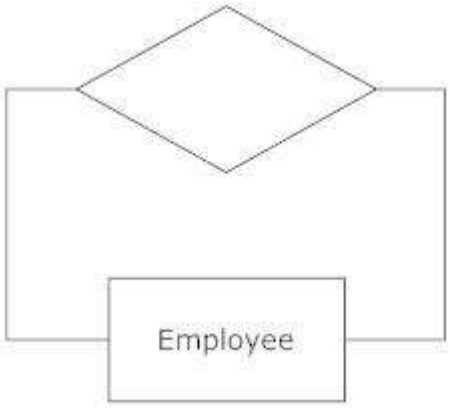
Description: An Entity Relationship (ER) Diagram is a type of flowchart that illustrates how "entities" such as people, objects or concepts relate to each other within a system. ER Diagrams are most often used to design or debug relational databases in the fields of software engineering, business information systems, education and research.

Components: An ER diagram is a means of visualizing how the information a system produces is related. There are five main components of an ERD:

Entities, which are represented by rectangles. An entity is an object or concept about which you want to store information. A weak entity is an entity that must defined by a foreign key relationship with another entity as it cannot be uniquely identified by its own attributes alone.



Actions, which are represented by diamond shapes, show how two entities share information in the database. In some cases, entities can be self-linked. For example, employees can supervise other employees.



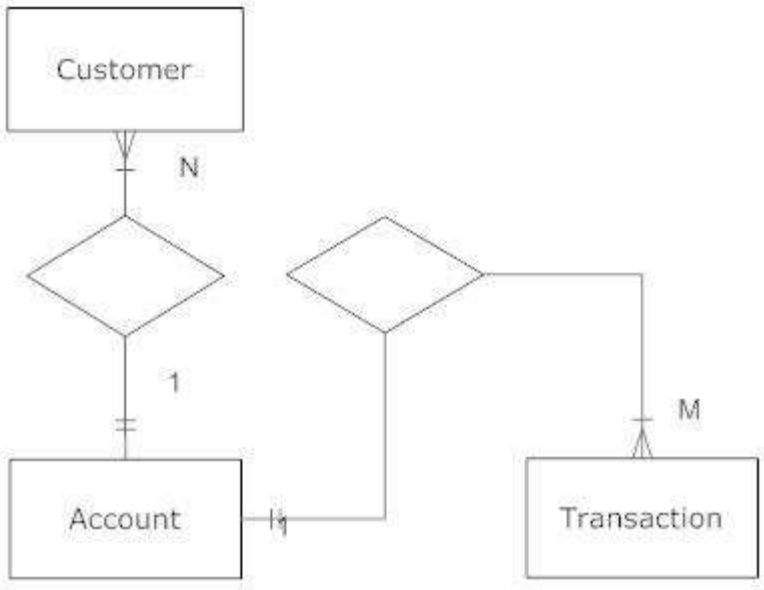
Attributes, which are represented by ovals. A key attribute is the unique, distinguishing characteristic of the entity. For example, an employee's social security number might be the employee's key attribute.

A multivalued attribute can have more than one value. For example, an employee entity can have multiple skill values.

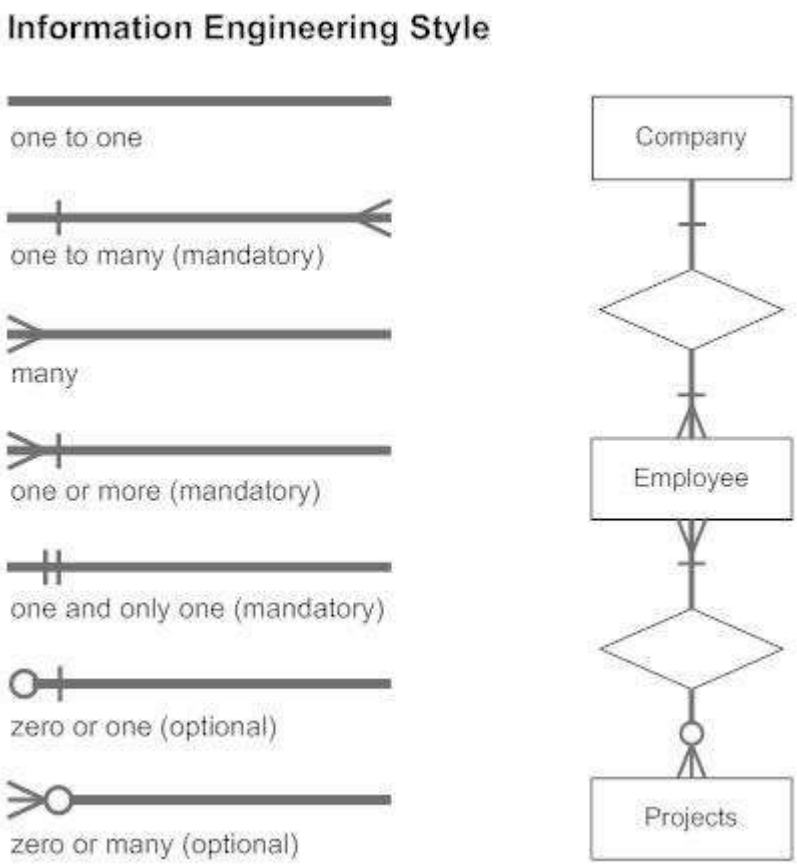
A derived attribute is based on another attribute. For example, an employee's monthly salary is based on the employee's annual salary.

Connecting lines, solid lines that connect attributes to show the relationships of entities in the diagram.

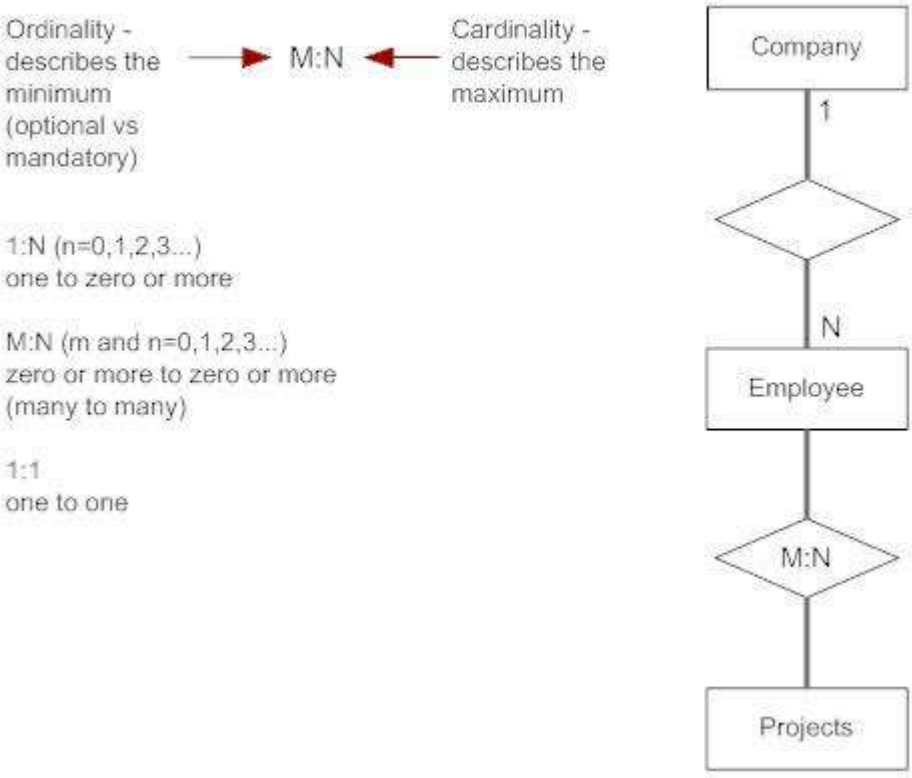
Cardinality specifies how many instances of an entity relate to one instance of another entity. Ordinality is also closely linked to cardinality. While cardinality specifies the occurrences of a relationship, ordinality describes the relationship as either mandatory or optional. In other words, cardinality specifies the maximum number of relationships and ordinality specifies the absolute minimum number of relationships.



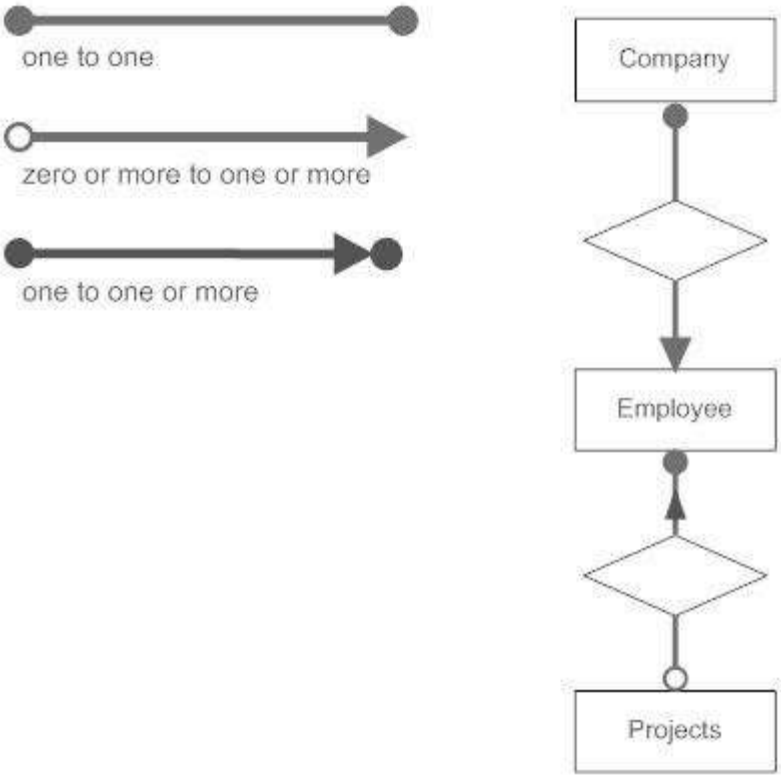
There are many notation styles that express cardinality.



Chen Style



Bachman Style



Tips for Effective ER Diagrams

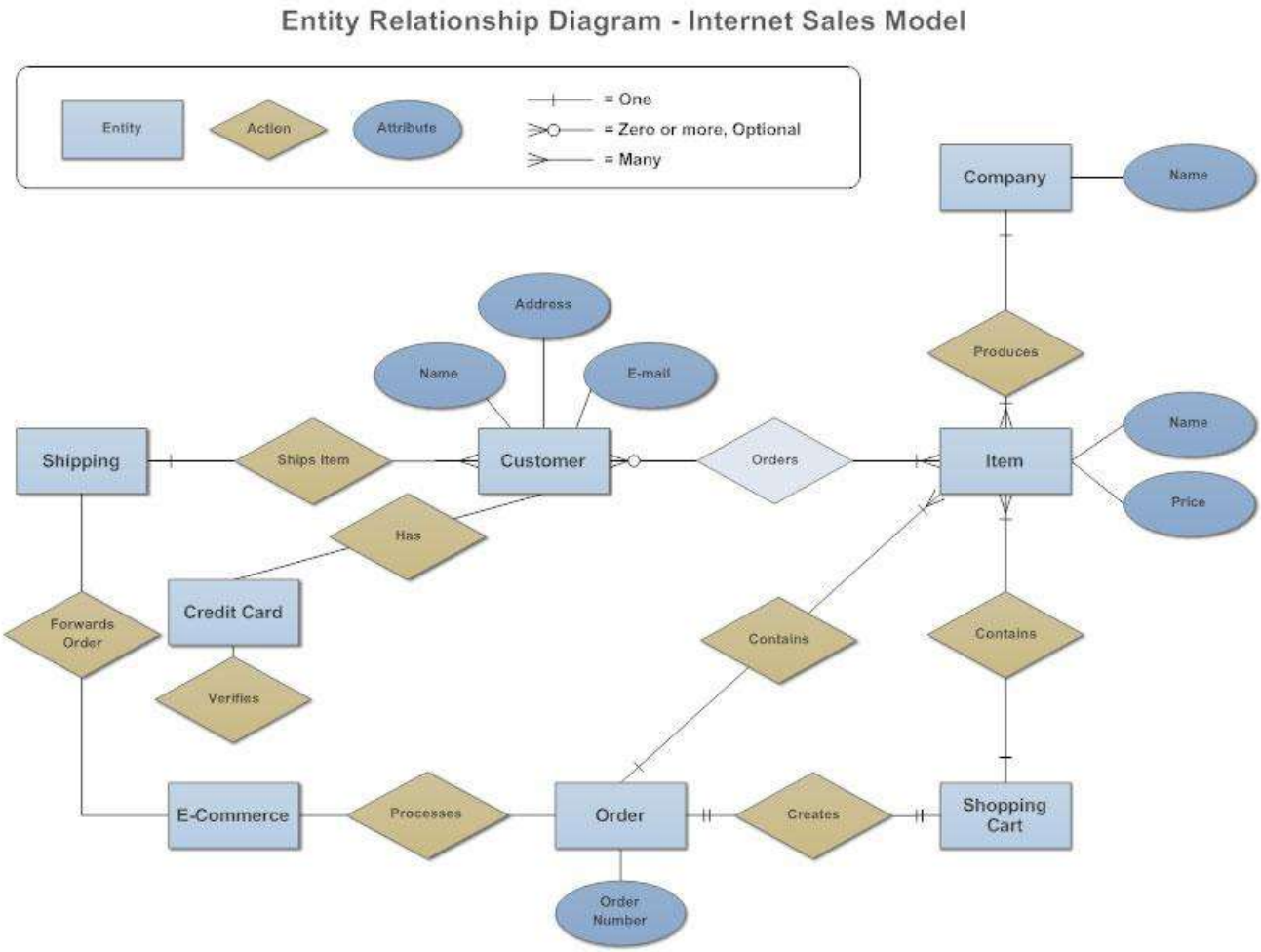
1. Make sure that each entity only appears once per diagram.

2. Name every entity, relationship, and attribute on your diagram.

3. Examine relationships between entities closely. Are they necessary? Are there any relationships missing? Eliminate any redundant relationships. Don't connect relationships to each other.

4. Use colors to highlight important portions of your diagram.

Entity Relationship Diagram Example



Practical Number: 3

Experiment Name: Data Flow Diagram

Hardware Requirements: Pentium 4 processor (2.4 GHz), 128 Mb RAM, Standard keyboard n mouse, colored monitor.

Software Requirements: SmartDraw/MS Word, Windows XP.

Outcome: Can produce the requirements in Data Flow diagram**.**

Objective: Prepare a Requirement document in by using Data Flow Diagram.

Description: Data flow diagram is graphical representation of flow of data in an information system. It is capable of depicting incoming data flow, outgoing data flow and stored data. The DFD does not mention anything about how data flows through the system.

There is a prominent difference between DFD and Flowchart. The flowchart depicts flow of control in program modules. DFDs depict flow of data in the system at various levels. DFD does not contain any control or branch elements.

Components:

External Entity an outside system that sends or receives data, communicating with the system being diagrammed. They are the sources and destinations of information entering or leaving the system. They might be an outside organization or person, a computer system or a business system. They are also known as terminators, sources and sinks or actors. They are typically drawn on the edges of the diagram**.**

Process any process that changes the data, producing an output. It might perform computations, or sort data based on logic, or direct the data flow based on business rules. A short label is used to describe the process, such as "Submit payment."

Data store files or repositories that hold information for later use, such as a database table or a membership form. Each data store receives a simple label, such as "Orders."

Data flow the route that data takes between the external entities, processes and data stores. It portrays the interface between the other components and is shown with arrows, typically labeled with a short data name, like "Billing details."

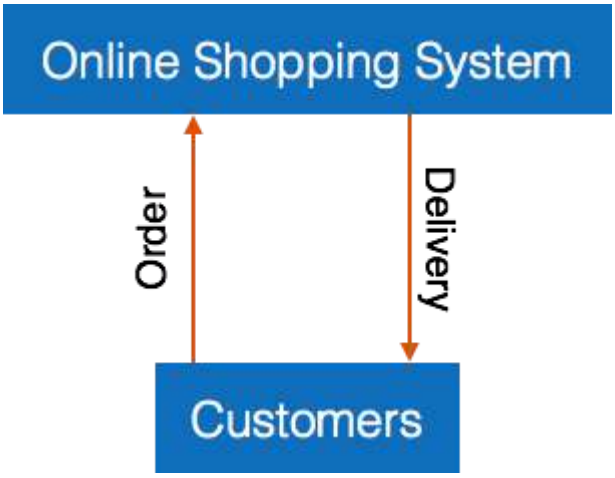
DFD rules

- Each process should have at least one input and an output.
- Each data store should have at least one data flow in and one data flow out.
- Data stored in a system must go through a process.
- All processes in a DFD go to another process or a data store.

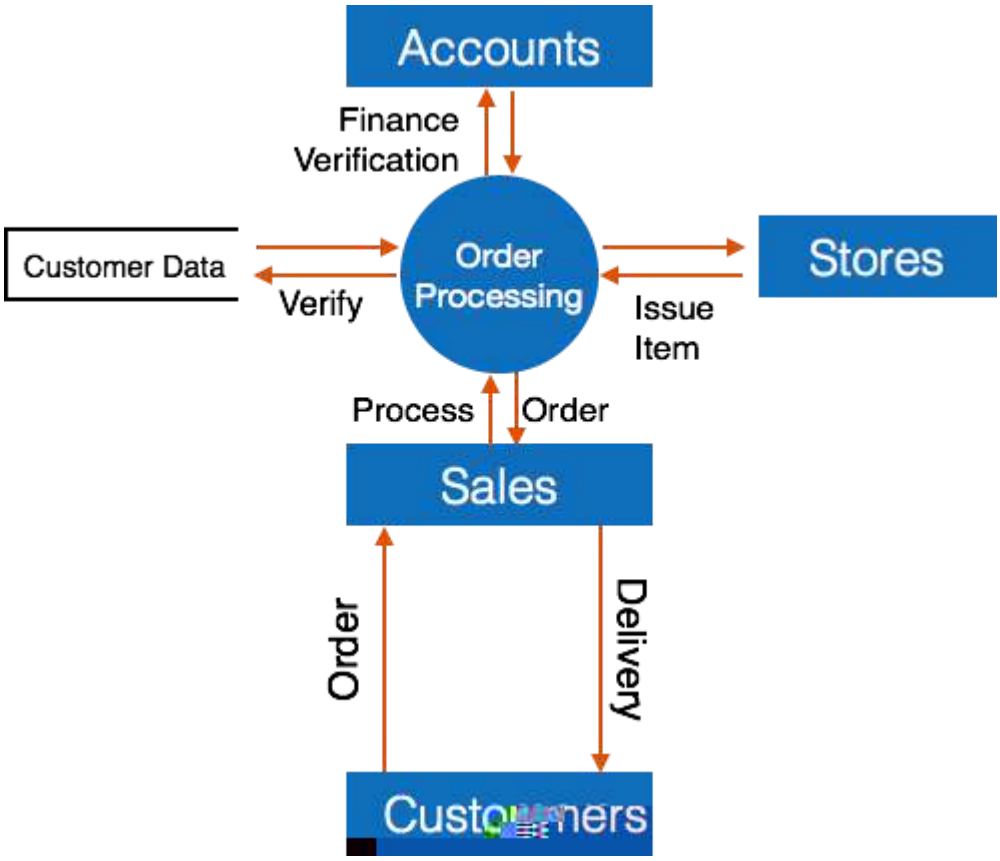
DFD Levels:

A data flow diagram can dive into progressively more detail by using levels and layers, zeroing in on a particular piece. DFD levels are numbered 0, 1 or 2, and occasionally go to even Level 3 or beyond.

1. DFD Level 0 is also called a Context Diagram. It's a basic overview of the whole system or process being analyzed or modeled. It's designed to be an at-a-glance view, showing the system as a single high-level process, with its relationship to external entities. It should be easily understood by a wide audience, including stakeholders, business analysts, data analysts and developers.



2. DFD Level 1 provides a more detailed breakout of pieces of the Context Level Diagram. You will highlight the main functions carried out by the system, as you break down the high-level process of the Context Diagram into its subprocesses.


Accounts
Finance
Verification
Customer Data
Verify
Order
Processing
Issue
Item
Stores
Process
Order
Sales
Order
Delivery
Customers

Practical Number: 4

Practical Name: Use Case Diagram

Hardware Requirements: Pentium 4 processor (2.4 GHz), 128 Mb RAM, Standard keyboard n mouse, colored monitor.

Software Requirements: Argo UML, Windows XP.

Outcome: Can produce the requirements in Use Case diagram**.**

Objective: Prepare a Requirement doccument in by using Use Case Diagram.

Description: According to the UML specification a use case diagram is “a diagram that shows the relationships among actors and use cases within a system.” Use case diagrams are often used to:

Provide an overview of all or part of the usage requirements for a system or organization in the form of an essential model or a business model

Communicate the scope of a development project

Model your analysis of your usage requirements in the form of a system use case model

Use case models should be developed from the point of view of your project stakeholders and not from the (often technical) point of view of developers. There are guidelines for:

Use Cases

Actors

Relationships

System Boundary Boxes

1. **Use Cases**

A use case describes a sequence of actions that provide a measurable value to an actor. A use case is drawn as a horizontal ellipse on a UML use case diagram.

1. Use Case Names Begin With a Strong Verb
2. Name Use Cases Using Domain Terminology
3. Place Your Primary Use Cases In The Top-Left Corner Of The Diagram
4. Imply Timing Considerations By Stacking Use Cases.

2. **Actors**

An actor is a person, organization, or external system that plays a role in one or more interactions with your system (actors are typically drawn as stick figures on UML Use Case diagrams).

1. Place Your Primary Actor(S) In The Top-Left Corner Of The Diagram
2. Draw Actors To The Outside Of A Use Case Diagram
3. Name Actors With Singular, Business-Relevant Nouns
4. Associate Each Actor With One Or More Use Cases
5. Actors Model Roles, Not Positions
6. Use <<system>> to Indicate System Actors
7. Actors Don't Interact With One Another
8. Introduce an Actor Called "Time" to Initiate Scheduled Events

3. **Relationships**

There are several types of relationships that may appear on a use case diagram:

An association between an actor and a use case
An association between two use cases
A generalization between two actors
A generalization between two use cases

Associations are depicted as lines connecting two modeling elements with an optional open-headed arrowhead on one end of the line indicating the direction of the initial invocation of the relationship. Generalizations are depicted as a close-headed arrow with the arrow pointing towards the more general modeling element.

1. Indicate An Association Between An Actor And A Use Case If The Actor Appears Within The Use Case Logic

2. Avoid Arrowheads On Actor-Use Case Relationships
3. Apply <<include>> When You Know Exactly When To Invoke The Use Case

4. Apply <<extend>> When A Use Case May Be Invoked Across Several Use Case Steps

5. Introduce <<extend>> associations sparingly

6. Generalize Use Cases When a Single Condition Results In Significantly New Business Logic

7. Do Not Apply <<uses>>, <<includes>>, or <<extends>>
8. Avoid More Than Two Levels Of Use Case Associations
9. Place An Included Use Case To The Right Of The Invoking Use Case
10. Place An Extending Use Case Below The Parent Use Case
11. Apply the “Is Like” Rule to Use Case Generalization
12. Place an Inheriting Use Case Below The Base Use Case
13. Apply the “Is Like” Rule to Actor Inheritance
14. Place an Inheriting Actor Below the Parent Actor

4. **System Boundary Boxes**

The rectangle around the use cases is called the system boundary box and as the name suggests it indicates the scope of your system – the use cases inside the rectangle represent the functionality that you intend to implement.

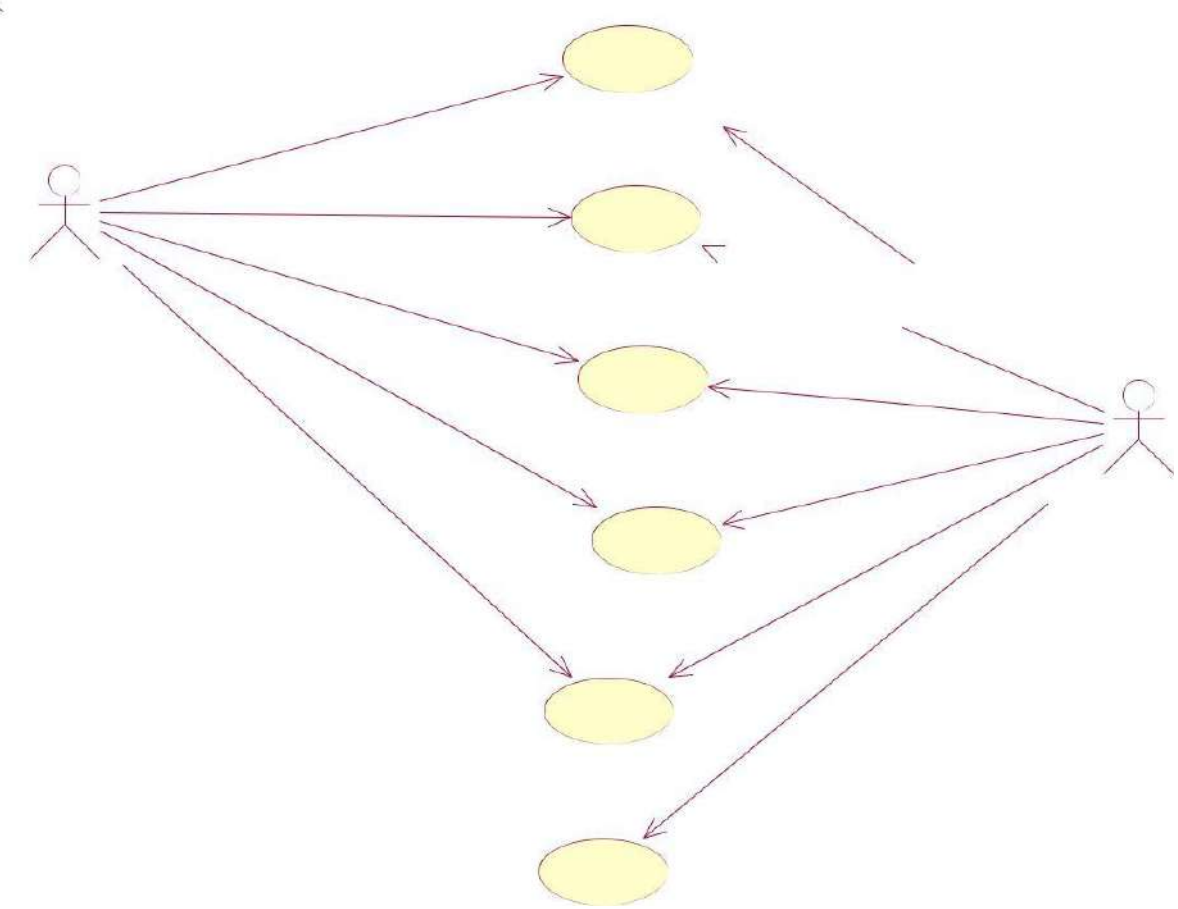
1. Indicate Release Scope with a System Boundary Box.
2. Avoid Meaningless System Boundary Boxes.

Creating Use Case Diagrams

we start by identifying as many actors as possible. You should ask how the actors interact with the system to identify an initial set of use cases. Then, on the diagram, you connect the actors with the use cases with which they are involved. If actor supplies information, initiates the use case, or receives any information as a result of the use case, then there should be an association between them.

Conclusion: The Use case diagram was made successfully by following the steps **described** above.

Output:



Data Transfer

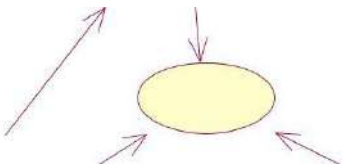
Administrator

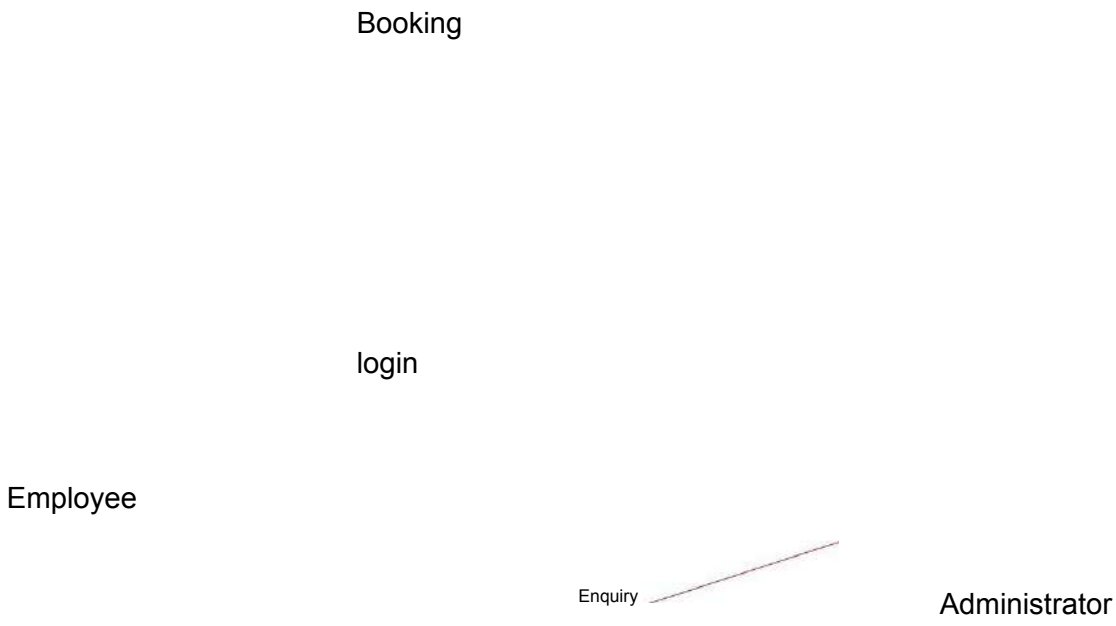
Mobility Management

Signalling Management Software Updation

Use Case Diagram

Cancellation of Booked Hall




Booking
login
Employee
Enquiry
Administrator
Report
Resources Update

Practical Number: 5

Practical Name: Activity Diagram

Experiment Name: Activity Diagram.

Outcome: Can produce the activity diagram for requirements modeling.

Objective: To Draw a sample activity diagram for real project or system.

Description:

Hardware Requirements:

Pentium 4 processor (2.4 GHz), 128 Mb RAM, Standard keyboard n mouse, colored monitor.

Software Requirements:

Argo UML, Windows XP,

Theory:

Activity diagrams are typically used for business process modeling, for modeling the logic captured by a single usecase or usage scenario, or for modeling the detailed logic of a business rule. Although UML activity diagrams could potentially model the internal logic of a complex operation it would be far better to simply rewrite the operation so that it is simple enough that you don't require an activity diagram. In many ways UML activity diagrams are the object-oriented equivalent of flow charts and data flow diagrams (DFDs) from structured development.

Let's start by describing the basic notation :

- **Initial node**. The filled in circle is the starting point of the diagram. An initial node isn't required although it does make it significantly easier to read the diagram.

- **Activity final node**. The filled circle with a border is the ending point. An activity diagram can have zero or more activity final nodes.

- **Activity**. The rounded rectangles represent activities that occur. An activity may be physical, such as *Inspect Forms*, or electronic, such as *Display Create Student Screen*.

- **Flow/edge**. The arrows on the diagram. Although there is a subtle difference between flows and edges,never a practical purpose for the difference although.

- **Fork**. A black bar with one flow going into it and several leaving it. This denotes the beginning of parallel activity.

- **Join**. A black bar with several flows entering it and one leaving it. All flows going into the join must reach it before processing may continue. This denotes the end of parallel processing.

- **Condition**. Text such as *[Incorrect Form]* on a flow, defining a guard which must evaluate to true in order to traverse the node.

- **Decision**. A diamond with one flow entering and several leaving. The flows leaving include conditions although some modelers will not indicate the conditions if it is obvious.

- **Merge**. A diamond with several flows entering and one leaving. The implication is that one or more incoming flows must reach this point until processing continues, based on any guards on the outgoing flow.

- **Partition**. If figure is organized into three partitions, it is also called swimlanes, indicating who/what is performing the activities (either the *Applicant*, *Registrar*, or *System*).

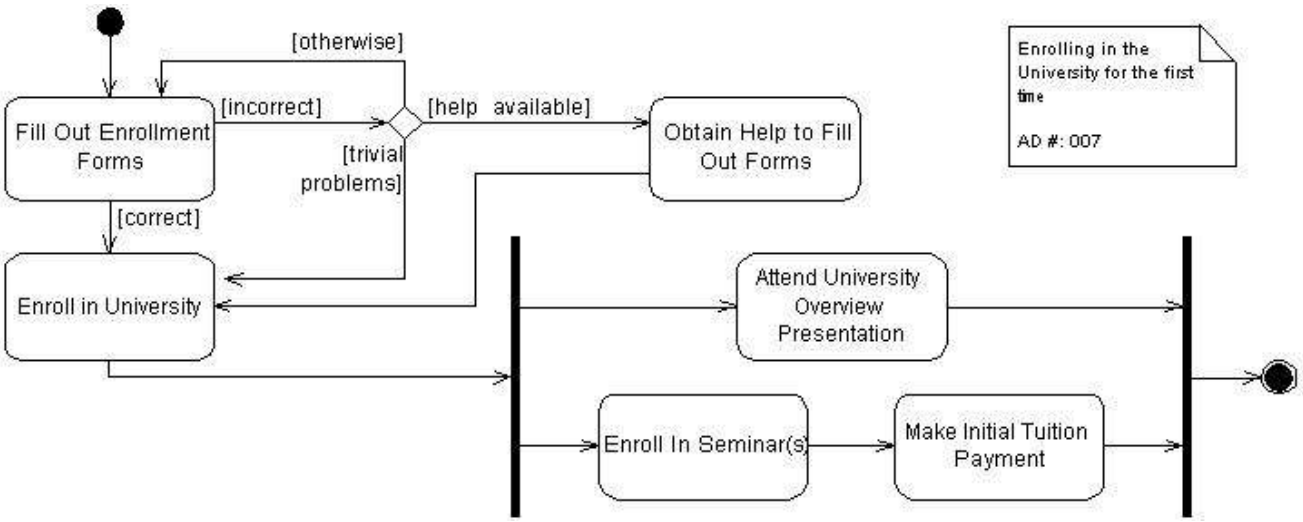
- **Sub-activity indicator**. The rake in the bottom corner of an activity, such as in the *Apply to University* activity, indicates that the activity is described by a more finely detailed activity diagram.

- **Flow final**. The circle with the X through it. This indicates that the process stops at this point.

GUIDELINES ASSOCIATED FOR DRAWING AN ACTIVITY DIAGRAM

General Guidelines
Decision Points
Decision Points
Guards
Parallel Activities
Swimlane Guidelines
.Action-Object Guidelines
General Guidelines

figure1. Modeling a business process with a UML Activity Diagram.



- Place The Start Point In The Top-Left Corner. A start point is modeled with a filled in circle, using the same notation that UML State Chart diagrams use. Every UML Activity Diagram should have a starting point, and placing it in the top-left corner reflects the way that people in Western cultures begin reading. Figure1, which models the business process of enrolling in a university, takes this approach.

- Always Include an Ending Point. An ending point is modeled with a filled in circle with a border around it, using the same notation that UML State Chart diagrams use. Figure1 is interesting because it does not include an end point because it describes a continuous process – sometimes the guidelines don't apply.

- Flowcharting Operations Implies the Need to Simplify. A good rule of thumb is that if an operation is so complex you need to develop a UML Activity diagram to understand it that you should consider refactoring it.

- **Activities**

An activity, also known as an activity state, on a UML Activity diagram typically represents the invocation of an operation, a step in a business process, or an entire business process.

- Question "Black Hole" Activities. A black hole activity is one that has transitions into it but none out, typically indicating that you have either missed one or more transitions.

- Question "Miracle" Activities. A miracle activity is one that has transitions out of it but none into it, something that should be true only of start points.

- **Decision Points**

A decision point is modeled as a diamond on a UML Activity diagram.

- Decision Points Should Reflect the Previous Activity. In figure1 we see that there is no label on the decision point, unlike traditional flowcharts which would include text describing the actual decision being made, we need to imply that the decision concerns whether the person was enrolled in the university based on the activity that the decision point follows. The guards, depicted using the format *[description]*, on the transitions leaving the decision point also help to describe the decision point.

- Avoid Superfluous Decision Points. The *Fill Out Enrollment Forms* activity in FIGURE1 includes an implied decision point, a check to see that the forms are filled out properly, which simplified the diagram by avoiding an additional diamond.

- **Guards**

A guard is a condition that must be true in order to traverse a transition.

- Each Transition Leaving a Decision Point Must Have a Guard

- Guards Should Not Overlap. For example guards such as x <0, x = 0, and x > 0 are consistent whereas guard such as x <= 0 and x >= 0 are not consistent because they overlap – it isn't clear what should happen when x is 0.

- Guards on Decision Points Must Form a Complete Set. For example, guards such as x < 0 and x >0 are not complete because it isn't clear what happens when x is

- Exit Transition Guards and Activity Invariants Must Form a Complete Set. An activity invariant is a condition that is always true when your system is processing an activity.

- Apply a [Otherwise] Guard for "Fall Through" Logic.

- Guards Are Optional. It is very common for a transition to not include a guard, even when an activity includes several exit transitions.

5. Parallel Activities

It is possible to show that activities can occur in parallel, as you see in FIGURE 1 depicted using two parallel bars. The first bar is called a fork, it has one transition

entering it and two or more transitions leaving it. The second bar is a join, with two or more transitions entering it and only one leaving it.

- A Fork Should Have a Corresponding Join. In general, for every start (fork) there is an end (join). In UML 2 it is not required to have a join, but it usually makes sense.

- Forks Have One Entry Transition.
- Joins Have One Exit Transition

- Avoid Superfluous Forks. FIGURE 2 depicts a simplified description of the software process of enterprise architectural modeling, a part of the Enterprise Unified Process (EUP). There is significant opportunity for parallelism in this process, in fact all of these activities could happen in parallel, but forks were not introduced because they would only have cluttered the diagram.

- **Swimlane Guidelines**

A swimlane is a way to group activities performed by the same actor on an activity diagram or to group activities in a single thread. FIGURE 2 includes three swimlanes, one for each actor.

Figure2. A UML activity diagram for the enterprise architectural modeling (simplified).

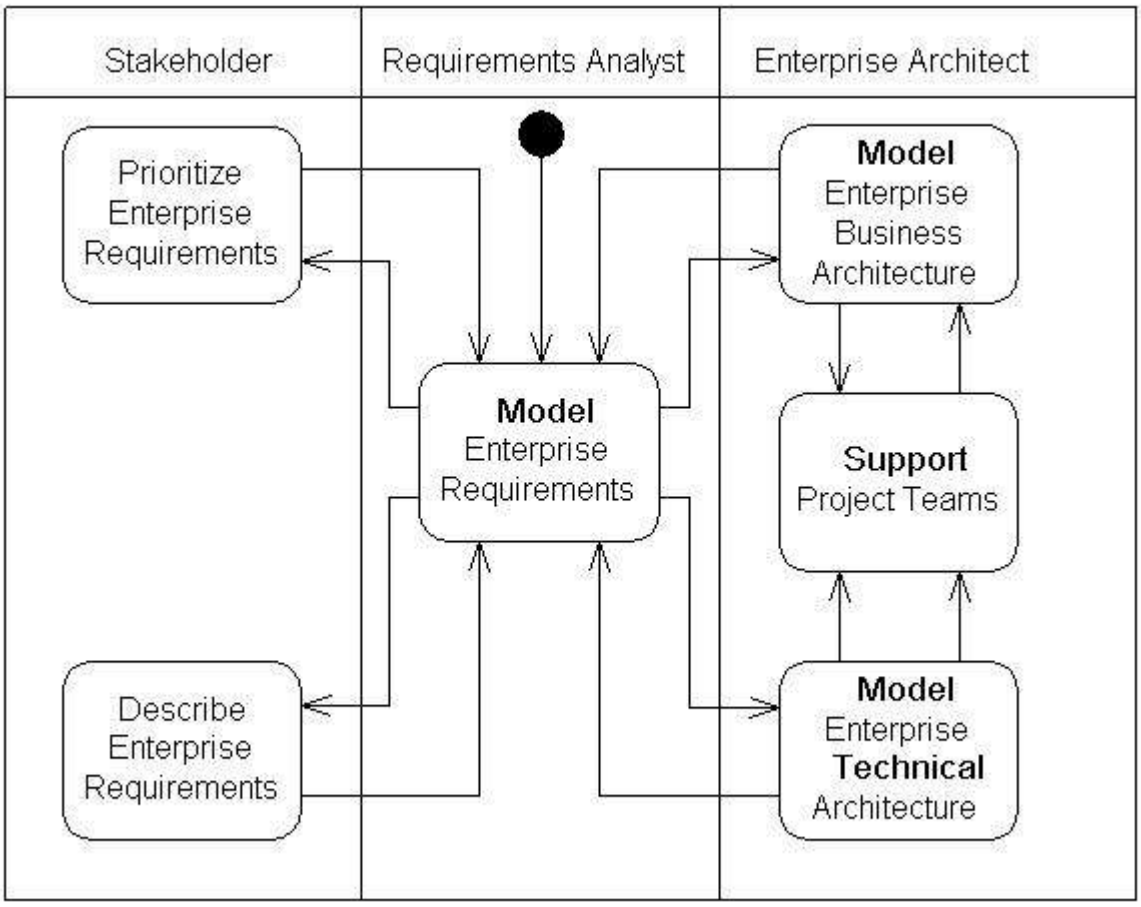
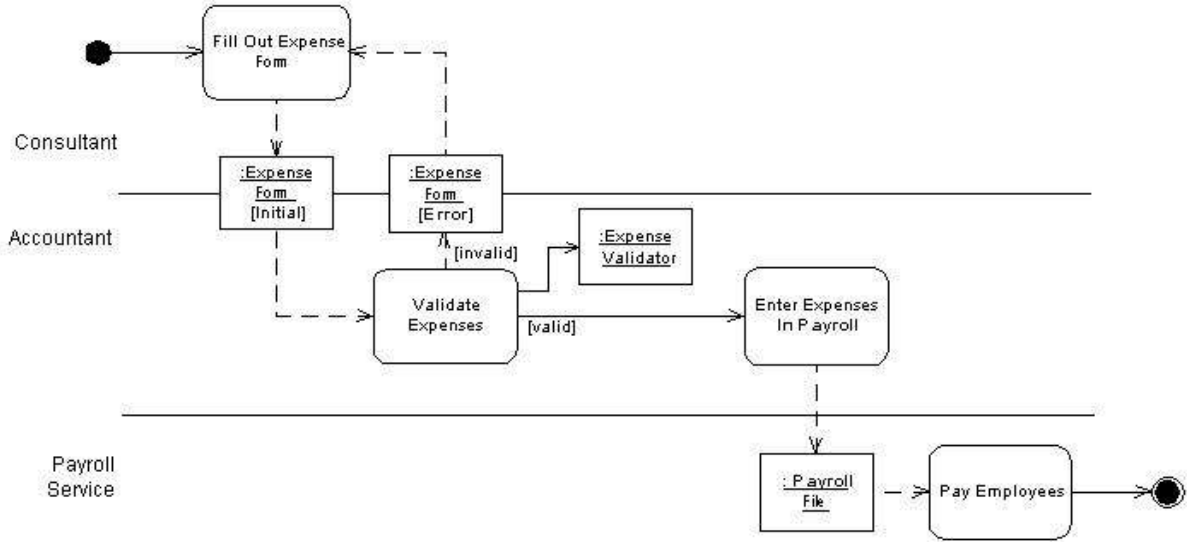


Figure 3. Submitting expenses.



- Order Swimlanes in a Logical Manner.

- Apply Swim Lanes To Linear Processes. A good rule of thumb is that swimlanes are best applied to linear processes, unlike the one depicted in FIGURE 3.

- Have Less Than Five Swimlanes.
- Consider Swimareas For Complex Diagrams.
- Swimareas Suggest The Need to Reorganize Into Smaller Activity Diagrams.

- Consider Horizontal Swimlanes for Business Processes. In FIGURE 3 you see that the swimlanes are drawn horizontally, going against common convention of drawing them vertically.

7 Action-Object Guidelines

Activities act on objects, In the strict object-oriented sense of the term an action object is a system object, a software construct. In the looser, and much more useful for business application modeling, sense of the term an action object is any sort of item. For example in FIGURE 3 the *ExpenseForm* action object is likely a paper form.

- Place Shared Action Objects on Swimlane Separators
- When An Object Appears Several Time Apply State Names
- State Names Should Reflect the Lifecycle Stage of an Action Object
- Show Only Critical Inputs and Outputs
- Depict Action Objects As Smaller Than Activities

Conclusion: The activity diagram was made successfully by following the steps **described** above.

Practical Number: 6

Experiment Name: Class Diagram

Experiment Name: Class diagram

Outcome: Class diagram helps to understand the static structure of a system. It shows relationships between classes, objects, attributes, and operations.

Objective: Identify the classes. Classify them as weak and strong classes and draw the class diagram.

Description:

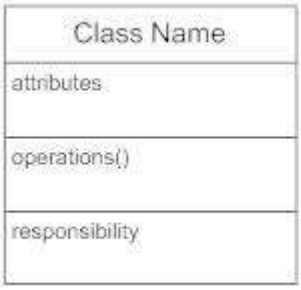
A class diagram models the static structure of a system. It shows relationships between classes, objects, attributes, and operations.

Basic Class Diagram Symbols and Notations

Classes

Classes represent an abstraction of entities with common characteristics. Associations represent the relationships between classes.

Illustrate classes with rectangles divided into compartments. Place the name of the class in the first partition (centered, bolded, and capitalized), list the attributes in the second partition (left-aligned, not bolded, and lowercase), and write operations into the third.



Class

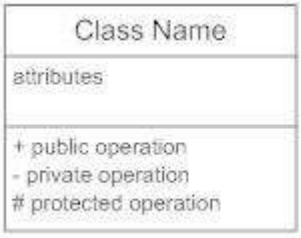
Active Classes

Active classes initiate and control the flow of activity, while passive classes store data and serve other classes. Illustrate active classes with a thicker border.

Active class

Visibility

Use visibility markers to signify who can access the information contained within a class. Private visibility, denoted with a - sign, hides information from anything outside the class partition. Public visibility, denoted with a + sign, allows all other classes to view the marked information. Protected visibility, denoted with a # sign, allows child classes to access information they inherited from a parent class.

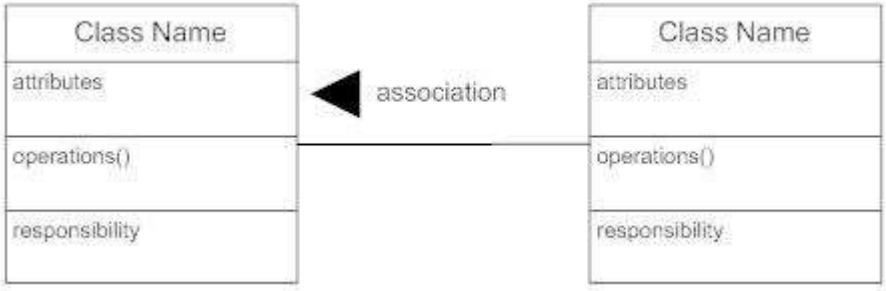


Visibility

Marker	Visibility
+	public
-	private
#	protected
~	package

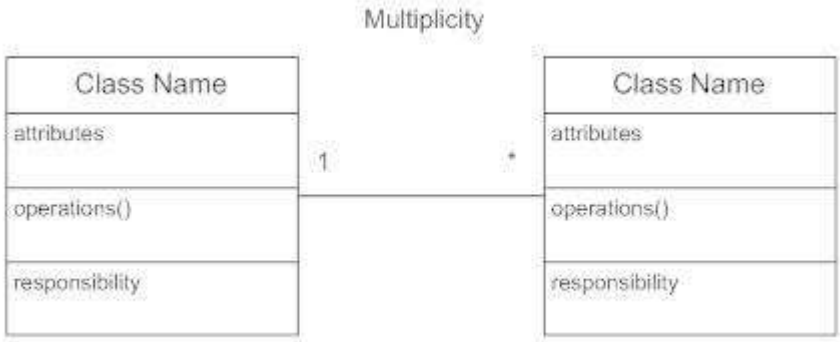
Associations

Associations represent static relationships between classes. Place association names above, on, or below the association line. Use a filled arrow to indicate the direction of the relationship. Place roles near the end of an association. Roles represent the way the two classes see each other.



Multiplicity (Cardinality)

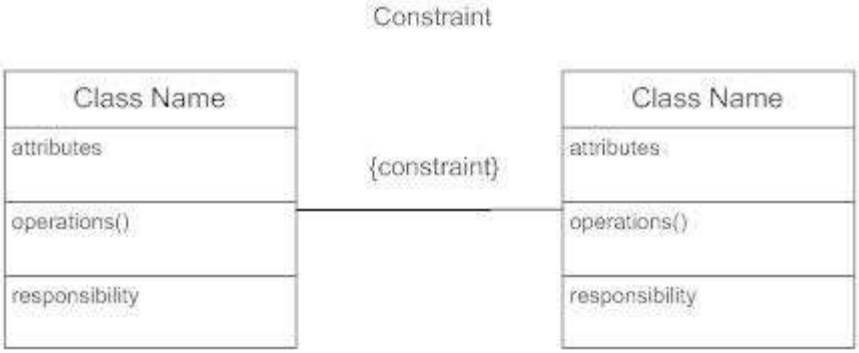
Place multiplicity notations near the ends of an association. These symbols indicate the number of instances of one class linked to one instance of the other class. For example, one company will have one or more employees, but each employee works for just one company.



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

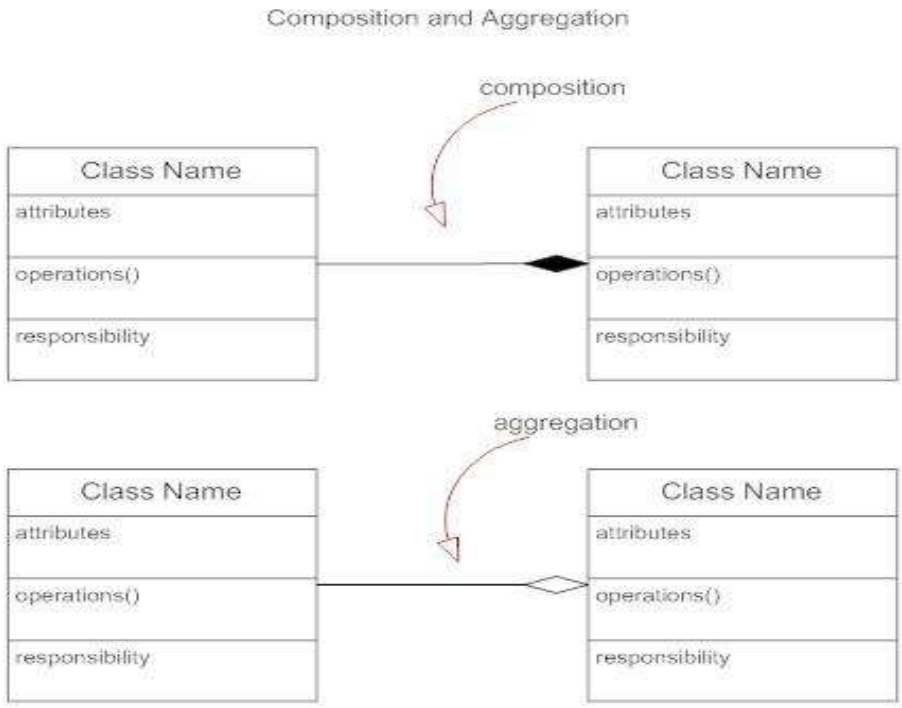
Constraint

Place constraints inside curly braces {}.



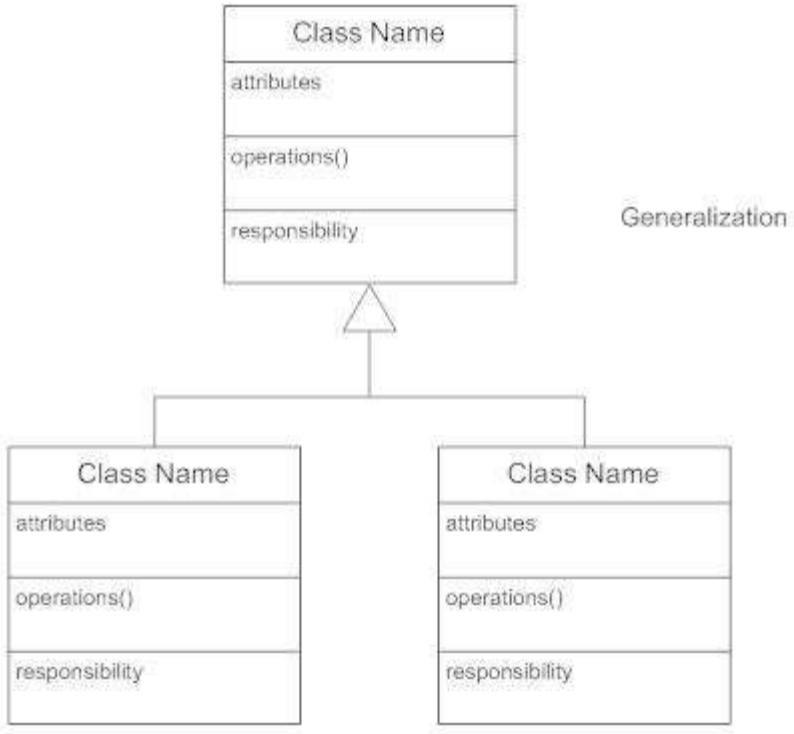
Composition and Aggregation

Composition is a special type of aggregation that denotes a strong ownership between Class A, the whole, and Class B, its part. Illustrate composition with a filled diamond. Use a hollow diamond to represent a simple aggregation relationship, in which the "whole" class plays a more important role than the "part" class, but the two classes are not dependent on each other. The diamond ends in both composition and aggregation relationships point toward the "whole" class (i.e., the aggregation).



Generalization

Generalization is another name for inheritance or an "is a" relationship. It refers to a relationship between two classes where one class is a specialized version of another. For example, Honda is a type of car. So the class Honda would have a generalization relationship with the class car.



In real life coding examples, the difference between inheritance and aggregation can be confusing. If you have an aggregation relationship, the aggregate (the whole) can access only the PUBLIC functions of the part class.

On the other hand, inheritance allows the inheriting class to access both the PUBLIC and PROTECTED functions of the superclass.

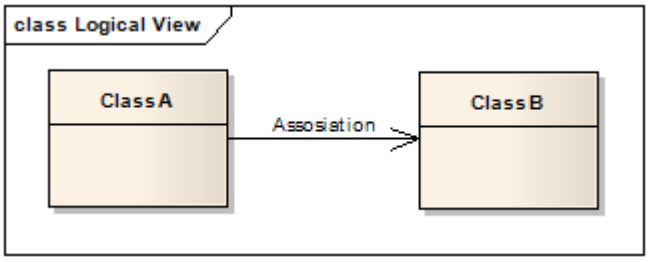
UML Class Diagram: Association, Aggregation and Composition

The UML Class diagram is used to visually describe the problem domain in terms of types of objects (classes) related to each other in different ways.

There are 3 primary inter-object relationships**: *Association*, *Aggregation***, and ***Composition***. Using the right relationship line is important for placing implicit restrictions on the visibility and propagation of changes to the related classes, a matter which plays an important role in understanding and reducing system complexity.

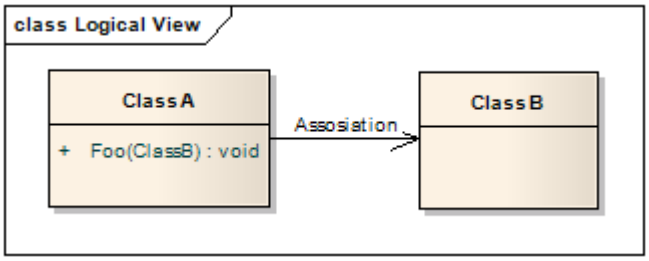
i) Association

The most abstract way to describe static relationship between classes is using the **Association** link, which simply states that there is some kind of a link or a dependency between two classes or more.



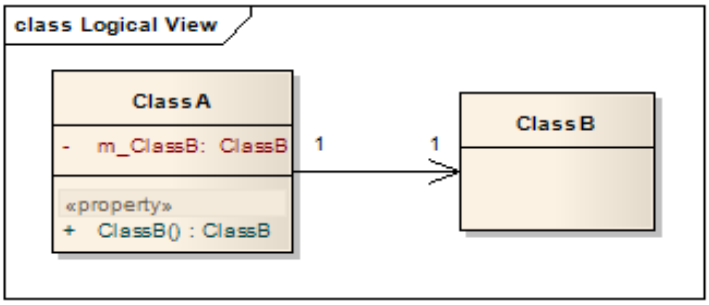
Weak Association

ClassA may be linked to ClassB in order to show that one of its methods includes parameter of ClassB instance, or returns instance of ClassB.



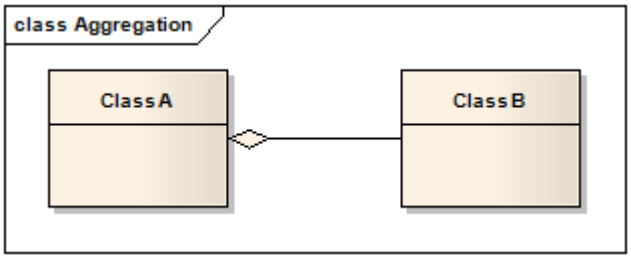
Strong Association

ClassA may also be linked to ClassB in order to show that it holds a reference to ClassB instance.

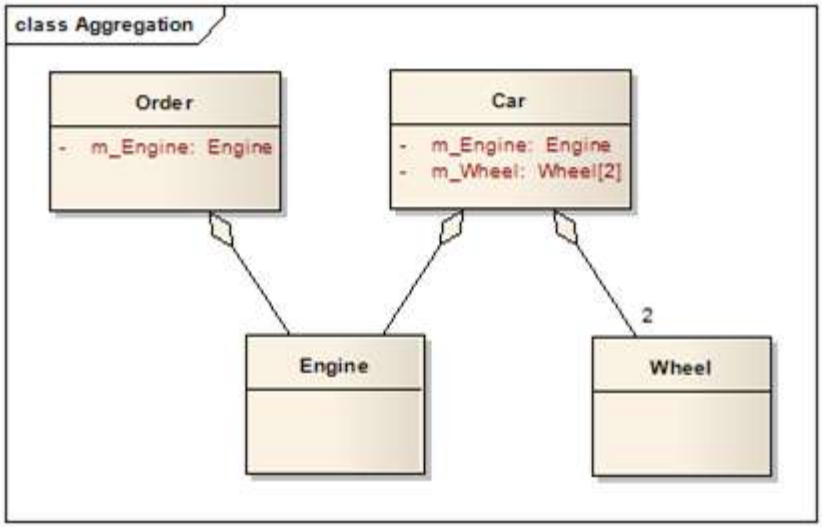


ii) Aggregation (Shared Association) (Weak Class)

In cases where there's a part-of relationship between ClassA (whole) and ClassB (part), we can be more specific and use the aggregation link instead of the association link, highlighting that the same ClassB instance can also be aggregated by other classes in the application (therefore aggregation is also known as shared association). Class B is weak Class.



It's important to note that the aggregation link doesn't state in any way that ClassA owns ClassB nor that there's a parent-child relationship (when parent deleted all its child's are being deleted as a result) between the two. Actually, quite the opposite! The aggregation link is usually used to stress the point that ClassA instance is not the exclusive container of ClassB instance, as in fact the same ClassB instance has another container/s.



Aggregation v.s. Association

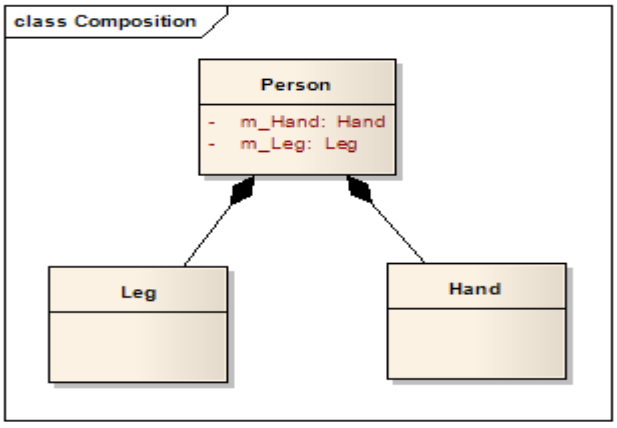
The association link can replace the aggregation link in every situation, while aggregation cannot replace association in situations where there's only a 'weak link'

between the classes, i.e. ClassA has method/s that contain parameter of ClassB, but ClassA doesn't hold reference to ClassB instance.

Martin Fowler suggest that the aggregation link should not be used at all because it has no added value and it disturb consistency, Quoting Jim Rumbaugh "Think of it as a modeling placebo".

iii) Composition (Not-Shared Association) (Strong Class)

We should be more specific and use the composition link in cases where in addition to the part-of relationship between ClassA and ClassB - there's a strong lifecycle dependency between the two, meaning that when ClassA is deleted then ClassB is also deleted as a result. Class Person is strong class.



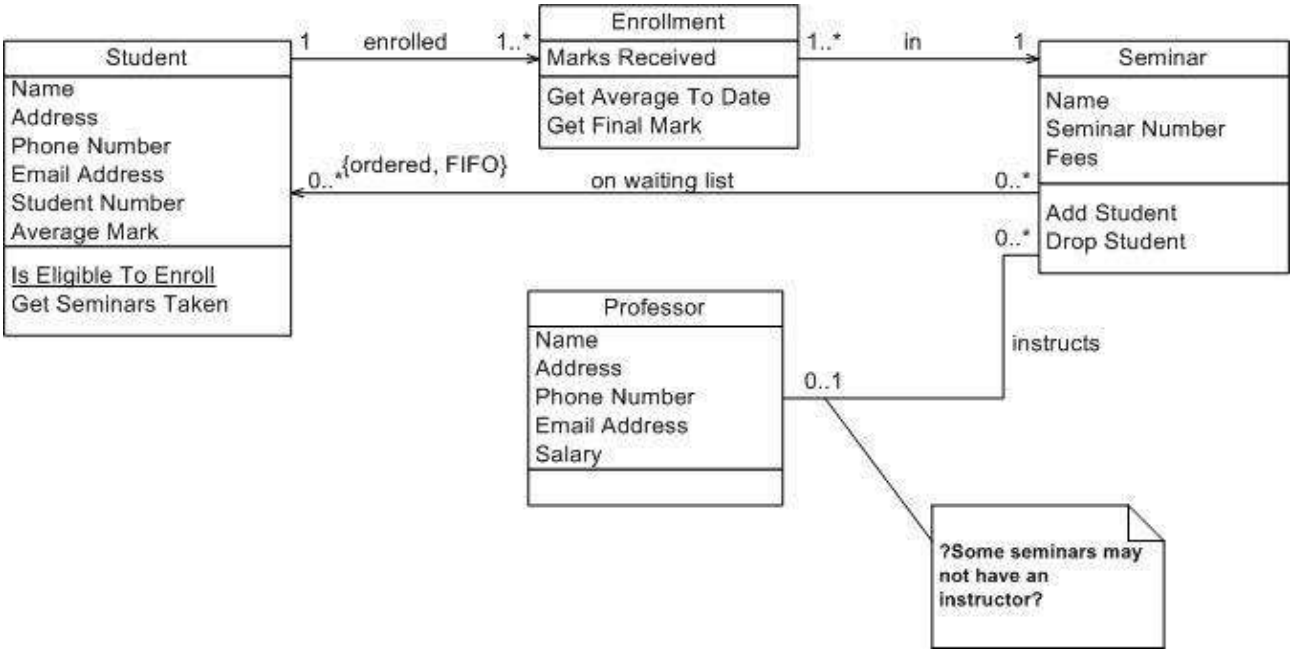
The composition link shows that a class (container, whole) has exclusive ownership over other class/s (parts), meaning that the container object and its parts constitute a parent-child/s relationship.

Unlike association and aggregation, when using the composition relationship, the composed class cannot appear as a return type or parameter type of the composite class. Thus, changes to the composed class cannot propagate to the rest of the system. Consequently, usage of composition limits complexity growth as the system grows.

Clarification: It is possible for a class to be composed by more than one class. For example, ClassA may be composed by ClassB and ClassC. However, unlike aggregation, instances of ClassB and ClassC will never share the same ClassA instance. That would violate the *propagation of changes* principle. ClassB instance will have its own instance of ClassA, and ClassC instance will have its own instance of ClassA.

Conclusion: The Class diagram was made successfully by following the steps **described** above.

Output: Class diagram



Practical Number: 7

Experiment Name: Component Diagram

Experiment Name: Component diagram.

Outcome: Can draw the Component diagram.

Objective: Drawing the component diagram

Description: A component is something required to execute a stereotype function. Examples of stereotypes in components include executables, documents, database tables, files, and library files.

Components are wired together by using an *assembly connector* to connect the required interface of one component with the provided interface of another component. This illustrates the *service consumer - service provider* relationship between the two components.

An *assembly connector* is a "connector between two components that defines that one component provides the services that another component requires. An assembly connector is a connector that is defined from a required interface or port to a provided interface or port."

When using a component diagram to show the internal structure of a component, the provided and required interfaces of the encompassing component can delegate to the corresponding interfaces of the contained components.

A *delegation connector* is a "connector that links the external contract of a component (as specified by its ports) to the internal realization of that behavior by the component's parts."[1]

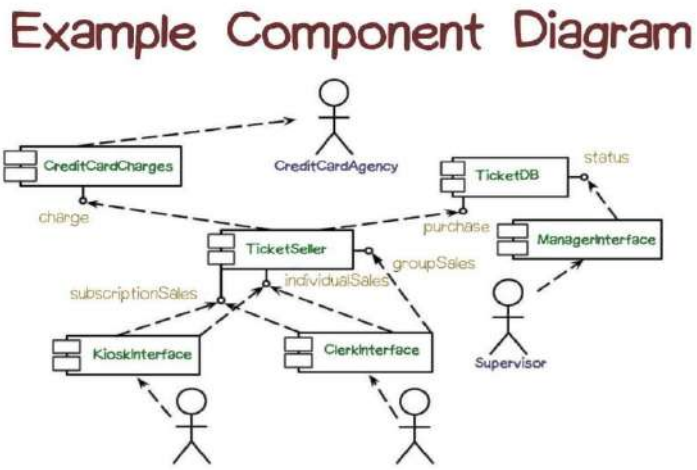
The example above illustrates what a typical insurance policy administration system might look like. Each of the components depicted in the above diagram may have other component diagrams illustrating its internal structure.

component is represented by a rectangle with either the keyword "component" or a stereotype in the top right corner: a small rectangle with two even smaller rectangles jutting out on the left.

The lollipop, a small circle on a stick, represents an implemented or provided interface. The socket symbol is a semicircle on a stick that can fit around the lollipop. This socket is a dependency or needed interface.

The component diagram notation set now makes it one of the easiest UML diagrams to draw. Figure 1 shows a simple component diagram using the former UML 1.4 notation; the example shows a relationship between two components: an Order System component that uses the Inventory System component. As you can see, a

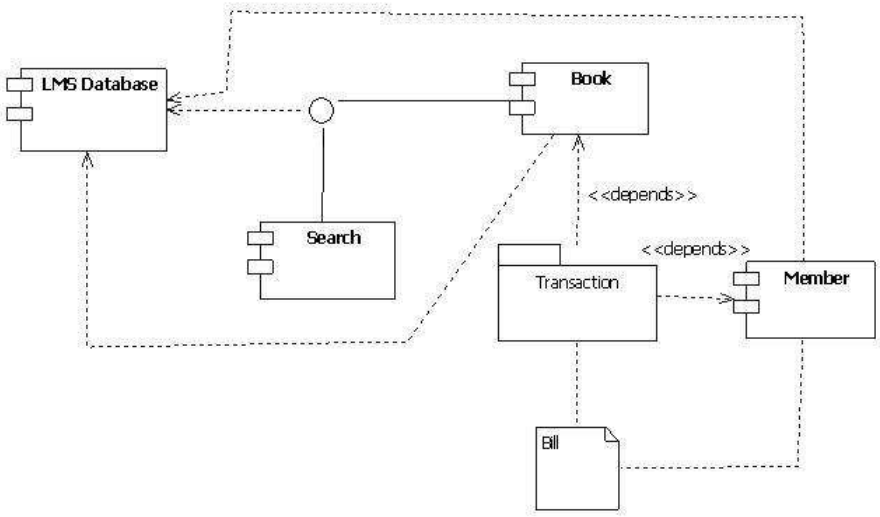
component in UML 1.4 was drawn as a rectangle with two smaller rectangles obtruding from its left side.



Conclusion: The Component diagram was made successfully by following the steps **described** above.

Output:

Component diagram for Library Management System:



Practical Number: 8

Experiment Name: Sequence Diagram

Experiment Name: Sequence diagram.

Outcome: Can draw the sequence diagram.

Objective: Drawing the sequence diagram

Hardware Requirements:

Pentium 4 processor (2.4 GHz), 128 Mb RAM, Standard keyboard and mouse, colored monitor.

Software Requirements: Argo UML, Windows XP

Theory:

UML sequence diagrams model the flow of logic within the system in a visual manner, enabling the user both to document and validate the logic, and are commonly used for both analysis and design purposes. Sequence diagrams are the most popular UML artifact for dynamic modeling, which focuses on identifying the behavior within your system. Sequence diagrams, along with class diagrams and physical data models are the most important design-level models for modern application development.

Sequence diagrams are typically used to model:

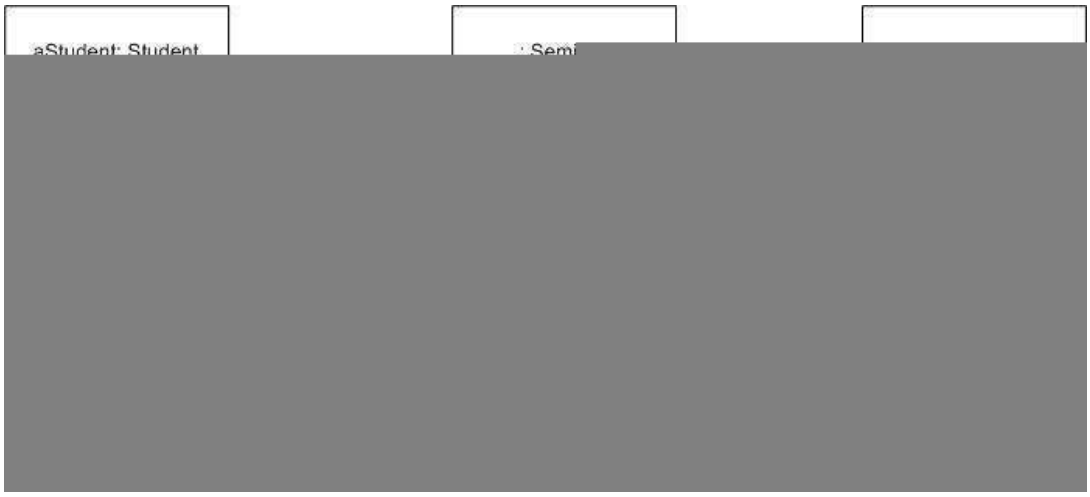
1. **Usage scenarios**. A usage scenario is a description of a potential way the system is used. The logic of a usage scenario may be part of a use case, perhaps an alternate course. It may also be one entire pass through a use case, such as the logic described by the basic course of action or a portion of the basic course of action, plus one or more alternate scenarios. The logic of a usage scenario may also be a pass through the logic contained in several use cases. For example, a student enrolls in the university, and then immediately enrolls in three seminars.

2. **The logic of methods**. Sequence diagrams can be used to explore the logic of a complex operation, function, or procedure. One way to think of sequence diagrams, particularly highly detailed diagrams, is as visual object code.

3. **The logic of services**. A service is effectively a high-level method, often one that can be invoked by a wide variety of clients. This includes web-services as well as business transactions implemented by a variety of technologies such as CICS/COBOL or CORBA-compliant object request brokers (ORBs).

FIG .shows the logic for how to enroll in a seminar. One should often develop a system-level sequence diagram to help both visualize and validate the logic of a usage scenario. It also helps to identify significant methods/services, such as checking to see if the applicant already exists as a student, which the system must support.

Figure 3. Enrolling in a seminar (method).



The dashed lines hanging from the boxes are called object lifelines, representing the life span of the object during the scenario being modeled. The long, thin boxes on the lifelines are activation boxes, also called method-invocation boxes, which indicate processing is being performed by the target object/class to fulfill a message.

How to Draw Sequence Diagrams

Sequence diagramming really is visual coding, even when you are modeling a usage scenario via a system-level sequence diagram.

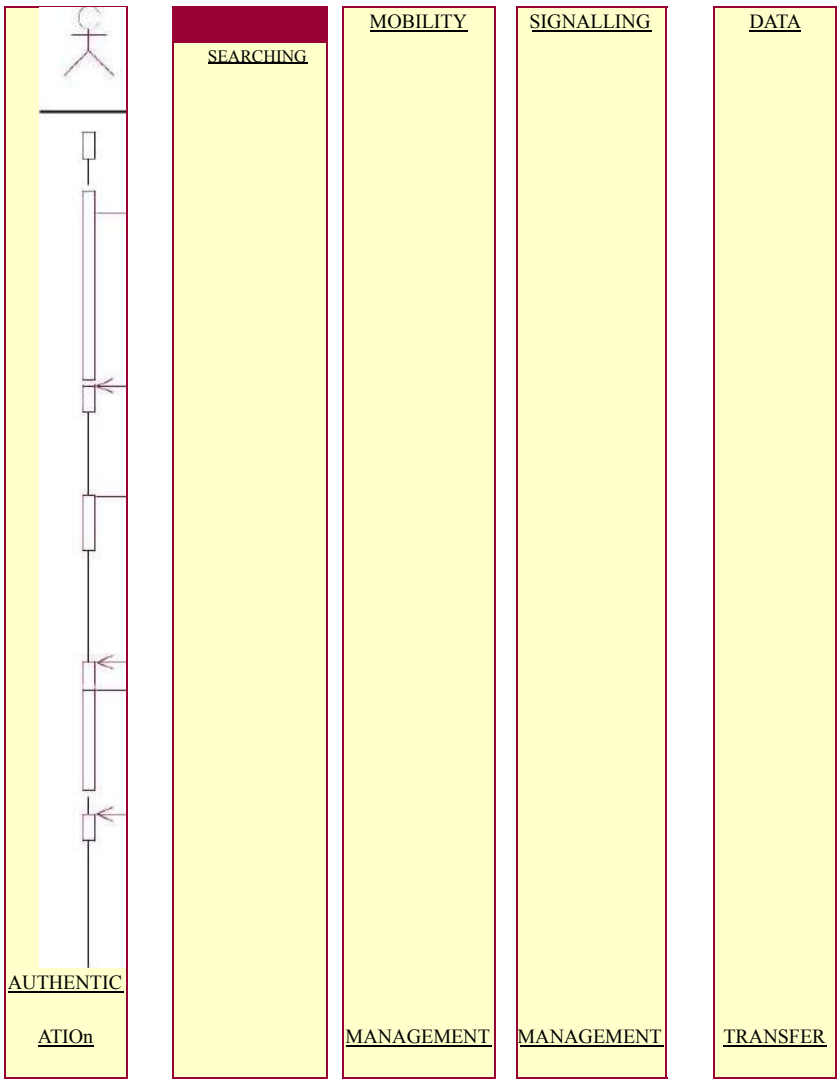
While creating a sequence diagram ,start by identifying the scope of what you are trying to model.You should typically tackle small usage scenarios at the system level or a single method/service at the detailed object level.

You should then work through the logic with at least one more person, laying out classifiers across the top as you need them. . The heart of the diagram is in the messages, which you add to the diagram one at a time as you work through the logic. You should rarely indicate return values, instead you should give messages intelligent names which often make it clear what is being returned.

It is interesting to note that as you sequence diagram you will identify new responsibilities for classes and objects, and, sometimes, even new classes. The implication is that you may want to update your class model appropriately, agile modelers will follow the practice *Create Several Models in Parallel*, something that CASE tools will do automatically. Remember, each message sent to a class invokes a static method/operation on that class each message sent to an object invokes an operation on that object.

Regarding style issues for sequence diagramming, prefer drawing messages going from left-to-right and return values from right-to-left, although that doesn't always work with complex objects/classes. Justify the label on messages and return values, so they are closest to the arrowhead. Also prefer to layer the sequence diagrams: from left-to-right. indicate the actors, then the controller class(es), and then the user interface class(es), and, finally, the business class(es). During design, you probably need to add system and persistence classes, which you should usually put on the right-most side of sequence diagrams. Laying your sequence diagrams in this manner often makes them easier to read and also makes it easier to find layering logic problems, such as user interface classes directly accessing persistence .

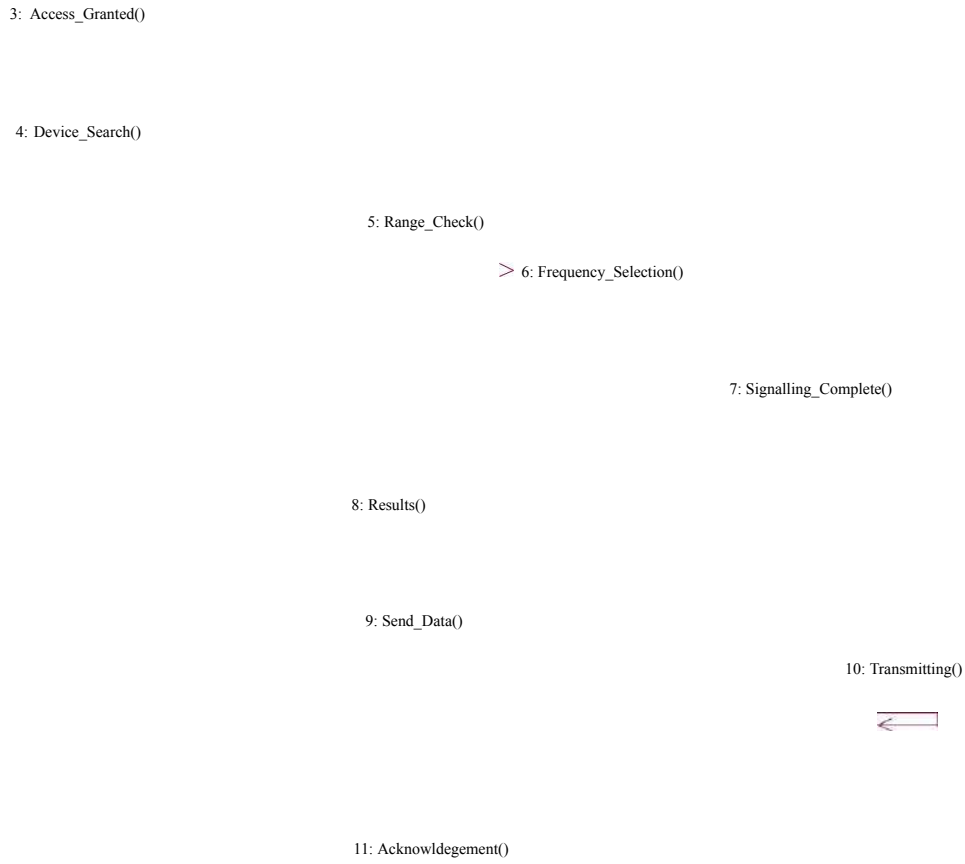
Another example of a sequence diagram



: User/BT

1: Access_ Request()

2: Authenticity_check()



Practical Number: 9

Experiment Name: Collboration Diagram

Experiment Name: Collaboration diagram .

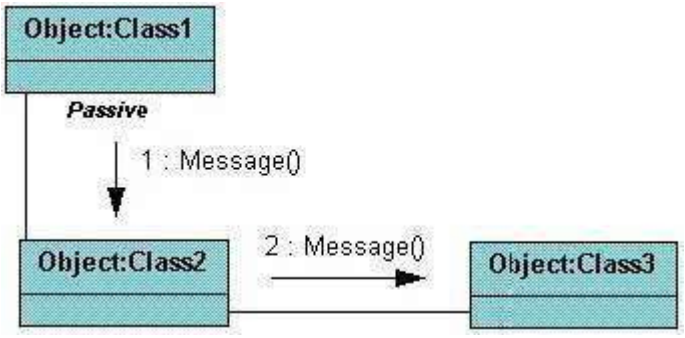
Outcome: Students will be able to draw the collaboration diagram

Objective: draw the collaboration diagram using argoUML.

Description:

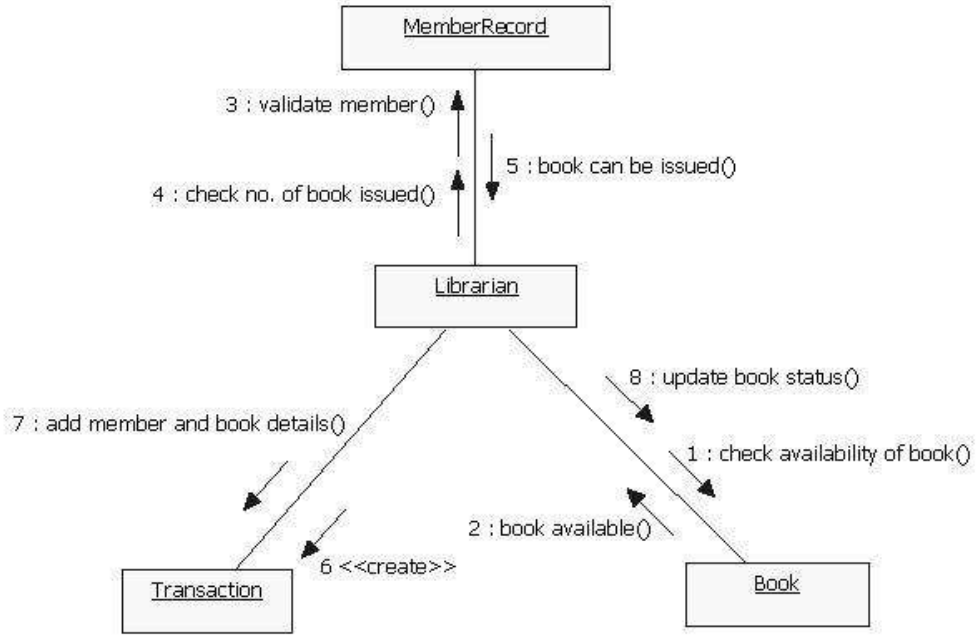
Collaboration diagrams ar relatively easy to draw. They show the relationship between objects and the order of messages passed between them. The objects are listed as icons and arrows indicate the messages being passed

between them. The numbers next to the messages are called sequence numbers. As the name suggests, they show the sequence of the messages as they are passed between the objects. There are many acceptable sequence numbering schemes in UML. A simple 1, 2, 3... format can be used, as the example below shows, or for more detailed and complex diagrams a 1, 1.1 ,1.2, 1.2.1... scheme can be used.

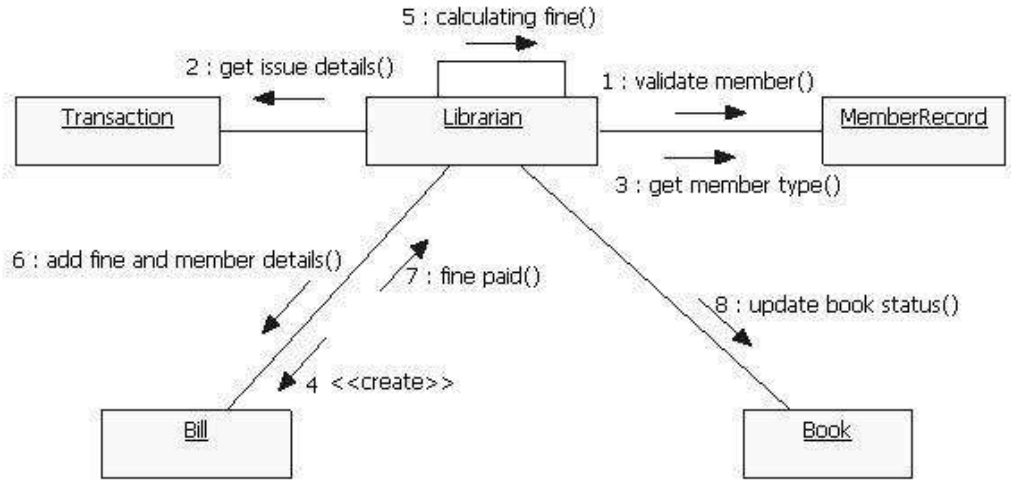


OUTPUT:

Collaboration diagram for issuing Book:



Collaboration diagram for returning Book:



Conclusion: The Collaboration diagram was made successfully by following the steps **described** above.

Withdrawal/Deposit: The software allows the user to select the kind of operation to be performed i.e. whether he wants to withdraw or deposit the money.

Amount:- The amount to be withdrawan or deposited is then mentioned by the user.

Denominations:- The user is also provided with the facility to mention the required denominations. Once he enters his requirements the machine goes through its calculations on the basis of current resources to check whether it is possible or not. If yes, the amount is given to the user otherwise other possible alternatives are displayed.

oney Deposition:- Money deposition shall be done with an envelope. After typing the amount to be deposited and verification of the same, the customer must insert the envelope in the depositary.
ance Transfer:- Balance transfer shall be facilitated between any two accounts linked to the card for example saving and checking account.

Billing:- Any transaction shall be recorded in the form of a receipt and the same would be dispensed to the customer. The billing procedures are handled by the billing module that enable user to choose whether he wants the printed statement of the transaction or just the updation in his account.

Balance Enquiry:- Balance enquiry for any account linked to the card shall be facilitated.
Cancelling:- The customer shall abort a transaction with the press of a Cancel key. For example on entering a wrong depositing amount. In addition the user can also cancel the entire session by pressing the abort key and can start a fresh session all over again.

Map locating other machines:- The machine also has a facility of displaying the map that marks the locations of other ATM machines of the same bank in the entire city.

Mobile Bills Clearings:- The machine also allows the user to clear off his pending mobile bills there only, if the name of his operator is mentioned there in the list. The machine displays the list of the companies supported by that bank to the user.

2.3 User Characteristics

There are different kind of users that will be interacting with the system. The intended user of the software are as follows:-

User A: A novice ATM customer. This user has little or no experience with electronic means of account management and is not a frequent user of the product. User A will find the product easy to use due to simple explanatory screens for each ATM function. He is also assisted by an intarctive teaching mechanism at every atep of the transaction, both with the help of visual and audio help sessions.

User B: An experienced customer. This user has used an ATM on several occasions before and does most of his account management through the ATM. There is only a little help session that too at the beginning of the session thus making the transaction procedure more faster.

Maintenance Personnel: A bank employee. This user is familiar with the functioning of the ATM. This user is in charge of storing cash into the ATM vault and repairing the ATM in case of malfunction. This user is presented with a different display when he logs in with the admninistrator's password and is provided with options different from that of normal user. He has the authority to change or restrict various features provided by the software in situations of repairing.

2.4 *Constraints*

The major constraints that the project has are as follows:-

The ATM must service at most one person at a time.

The number of invalid pin entries attempted must not exceed three. After three unsuccessful login attempts, the card is seized/blocked and need to be unlocked by the bank.

The simultaneous access to an account through both, the ATM and the bank is not supported.

The minimum amount of money a user can withdraw is Rs 100/- and the maximum amount of money a user can withdraw in a session is Rs.10,000/- and the maximum amount he can withdraw in a day is Rs 20,000/-

fore the transaction is carried out, a check is performed by the machine to ensure that a minimum amount of Rs 1000/- is left in the user's account after the withdrawal failing which the withdrawal is denied.

← The minimum amount a user can deposit is Rs 100/- and the maximum amount he can deposit is Rs 10,000/-.

A user can select only that cellular operator for mobile bill clearings that is supported by the bank.The software requires a minimum memory of 20GB. The databse used should be Oracle7.0.There shall be a printer installed with the machine to provide the use er with the printed statement of the transaction. For voice interactions, speakers should also be there to accompany the machine.

2.5 *Assumptions and Dependencies*

The requirements stated in the SRS could be affected by the following factors:

One major dependency that the project might face is the changes that need to be incorporated with the changes in the bank policies regarding different services. As the policies changes the system needs to be updated with the same immediately. A delay in doing the same will result to tremendous loss to the bank. So this should be changed as and when required by the developer.

2.6

Another constraint relating to the operating environment is that we are specific to Oracle Database.

2.7

The project could be largely affected if some amount is withdrawn from the user's account from the bank at the same time when someone is accessing that account through the ATM machine. Such a condition shall be taken care of.

2.8

At this stage no quantities measures are imposed on the software in terms of speed and memory although it is implied that all functions will be optimized with respect to speed and memory.

It is furthermore assumed that the scope of the package will increase considerably in the future.

3.1.1 User Interface Requirements

The interface provided to the user should be a very user-friendly one and it should provide an optional interactive help for each of the service listed. The interface provided is a menu driven one and the following screens will be provided:-

1. A login screen is provided in the beginning for entering the required username/pin no. and account number.

2. An unsuccessful login leads to a reattempt(maximum three) screen for again entering the same information. The successful login leads to a screen displaying a list of supported languagesfrom which a user can select any one.

3. In case of administrator, a screen will be shown having optins to reboot system, shut down system, block system, disable any service.

4. In case of reboot/ shut down, a screen is displayed to confirm the user's will to reboot and also allow the user to take any backup if needed.

5. In case of blocking system, a screen is provided asking for the card no. By entering the carnd no of a particular user, system accees can be blocked for him.

6. Administrator is also provided with a screen that enables him to block any service provided to the user by enterin the name of the service or by selecting it from the list displayed.

7. After the login, a screen with a number of options is then shown to the user. It contains all the options along with their brief description to enable the user to understand their functioning and select the proper option.

8. A screen will be provided for user to check his account balance.

9. A screen will be provided that displays the location of all other ATMs of same bank elsewhere in the city.

10. A screen will be provided for the user to perform various transactions in his account.

The following reports will be generated after each session dealt with in the machine:-

1. The login time and logout time along with the user's pin no and account number is registered in the bank's database.

2. The ATM's branch ID through which the session is established is also noted down in the bank's database.

3. Various changes in the user's account after the transactions,if any, are reported in the database.

4. A printed statement is generated for the user displaying all the transactions he performed.

Other various user interface requirements that need to be fulfilled are as follows:-

The display screen shall have 256 color resolution.

The display screen shall also support touchscreen facility.

The speakers shall support Yamaha codecs.

The keypad shall consist of 16 tactile keys.

There shall be 8 tactile function keys.

The keyboard will be weather resistant.

The transaction receipt shall be 3.1" × 6".

The statement receipt shall be 4.2" × 12".

The deposit envelopes shall be 9" long and 4" wide.

3.1.2 **Hardware Interface Requirements**

There are various hardware components with which the machine is required to interact. Various hardware interface requirements that need to be fulfilled for successful functioning of the software are as follows:-

The ATM power supply shall have a 10/220 V AC manual switch.

The ATM card should have the following physical dimensions:-

- Width - 85.47mm-85.72mm
- Height - 53.92mm-54.03mm
- Thickness - 0.76mm+0.08mm

The card reader shall be a magnetic stripe reader

The card reader shall have Smart card option.

The slot for a card in thye card reader may include an extra indentation for the embossed area of the card. In effect it acts as a polarization key and may be used to aid the correct insertion orientation of the card. This is an additional characteristic to the magnetic field sensor which operates off the magnetic stripe and is used to open a mechanical gate on devices such as ATMs.
There shall be a 40 column dot matrix receipt printer.

There shall be a 40 column dot matrix statement printer.

The receipt dispenser shall be a maximum of 4" width and 0.5" thickness.

The statement dispenser shall be a maximum of 5" width and 0.5" thickness.

The envelope depository shall be a maximum of 4.5" width, 10" length and 0.5" thickness.

Screen resolution of at least 800X600-required for proper and complete viewing of screens. Higher resolution would not be a problem.

Software Interface Requirements

In order to perform various different functions, this software needs to interact with various other softwares. So there are certain software interface requirements that need to be fulfilled which are listed as follows:-

The transaction management software used to manage the transaction and keep track of resources shall be BMS version 2.0.

The card management software used to verify pin no and login shall be CMS version 3.0.
Yamaha codecs 367/98 for active speakers.

The database used to keep record of user accounts shall be Oracle version7.0.

Communication Interface Requirements

The machine needs to communicate with the main branch for each session for various functions such as login verification, account access etc. so the following are the various communication interface requirements that are needed to be fulfilled in order to run the software successfully:-

The system will employ dial-up POS with the central server for low cost communication.
The communication protocol used shall be TCP/IP.

Protocol used for data transfer shall be File Transfer Protocol.(FTP)

4. ***System Features***

1. Remote Banking and Account Management

Description

The system is designed to provide the user with the facility of remote banking and perform various other functions at an interface without any aid of human bank teller. The functioning of the system shall be as follows:-

At the start, the user is provided with a log in screen and he is required to enter his PIN NO. and Account details which are then verified by the machine. In case of an unsuccessful attempt a user is asked again for his credentials but the maximum number of attempt given to the user is limited to 3 only, failing which his card is blocked and need to be unblocked by the bank for any future use.

After a successful log in, the user is presented with a list of language. The user can select any one in the list for interaction with the machine for the entire session. After the language selection the user is also asked whether he wants to fix that language for future use also so that he is never asked for language in future. In addition there is also a facility for the user to switch to any other language during that session.

After the language selection, the user is directed towards a main page that displays a set of options/services along with their brief description, enabling the user to understand their functioning. The user can select any of the listed option and can continue with the transaction.

The machine also provides the user with a number of miscellaneous services such as:

The machine lists a set of operators that are supported by the bank. A user can clear off his pending mobile phone bills be selecting his operator.

The machine also has the facility to display a map that marks the location of other ATMs of the same bank in the city. This may help the user to look for the ATM nearest to his destination.

At any moment if the user wants to abort the transaction, he is provided with an option to cancel it. Just by pressing the abort button he can cancel all the changes made so far and can begin with a new transaction.

After the user is finished with his work, for security purpose, he is required to log out and then take his card out of the slot.

Validity Checks

In order to gain access to the system, the user is required to enter his/her correct user id/pin no and account no failing which his card may be blocked.

The user can access only one account at a time and can enter only one account no.

Also if the user is an administrator, he is required to enter his login id in order to access and change the facilities provided by the system.

Sequencing Information

The information about the users and their account should be entered into the database prior to any of the transactions and the backup be maintained for all account information

Error Handling/ Response to Abnormal Situations

If any of the above validation/sequencing flow does not hold true, appropriate error messages will be prompted to the user for doing the needful.

2. Receipt Generation

After ech transaction user has performed, a receipt is generated that contains all the information about the transaction. The format of the generated receipt is as shown below:-

KPM BANK

Branch name/Id (address)

Login Time:- **Date:-**

Account No:-

User Name:-

TRANSACTIONS:

FROM
TO TYPE
AMOUNT

Logout Time:- **BARCODE**

Thank You For your visit.

See you soon.

Nonfunctional Requirements

5.1 Performance Requirements

The following list provides a brief summary of the performance requirements for the software:

5.1.1 Capacity

The ATM shall provide customers a 24 hour service.

5.1.2 Dynamic Requirements

The card verification time must not exceed 0.8 sec. under normal server workload and 1 sec. under peak server workload.

The pin number verification time must not exceed 0.3 sec. under normal server workload and 0.5 sec. under peak server workload.

Account balance display time must not exceed 2 sec. under normal server workload and 3 sec. under peak server workload.

Account balance transfer time must not exceed 3 sec. under normal server workload and 4 sec. under peak server workload.

Cash withdrawal transaction time must not exceed 4 sec. under normal server workload and 5 sec. under peak server workload.

Deposit transaction time after insertion of the deposit envelope must not exceed 5 sec. under normal server workload and 6 sec. under peak server workload.

Receipt printing time after must not exceed 3 sec. under normal server and peak server workload.

Touch screen and button response time must not exceed 5000ms.

Credit card advance time must not exceed 6 sec. under normal traffic and server and peak traffic and server workload.

5.1.3 Quality

The primary objective s to produce quality software. As the quality of a piece of software is difficult to measure quantitatively, the following guidelines will be used when judging the quality of the software:

5.2 Software Quality Attributes

5.2.1 Reliability

The data communication protocol shall be such that it ensures reliability and quality of data and voice transmission in a mobile environment. For example, CDMA.

The memory system shall be of non-volatile type.

5.2.2 Availability

The product will have a backup power supply incase of power failures.

Any abnormal operations shall result in the shutting down of the system.

After abnormal shutdown of the ATM, the system shall have to be manually restarted by a maintenance personnel.

There should be no inconsistency introduced in the account during whose transaction the system is abnormally shut down.

5.2.3 **Security**

The system shall be compatible with AIMS security standards.

The system shall have two levels of security i.e. ATM card and pin verification both authenticated by the CMS software.

The Encryption standard used during pin transmission shall be triple DES.
The password shall be 6-14 characters long.

Passwords shall not contain name of customers as they are easy to be hacked.

Passwords can contain digit, hyphen and underscore.

User should be provided with only three attempts for login failing which his card needs to be blocked.

There shall be a security camera installed near the ATM.

There shall be a secured cash vault with a combination locking system.

The product cabinet cover shall be manufactured using Fiber glass for security purposes.

5.2.4 **Maintainability**

The system components i.e. modem, memory, disk, drives shall be easily serviceable without requiring access to the vault.

The system should have the mechanism of self-monitoring periodically in order to detect any fault.

The system should inform the main branch automatically as soon as it detects any error. The kind of fault and the problem being encountered should also be mentioned by the system automatically.

5.3 ***Business Rules***

The business rules for the software are as follows:

The Administrator has the authority to fix the rules and regulations and to set or update the policies as and when required.
The staff at the bank performs the following:

a. Making the entries in the system regarding all the details of the bank account of the user.

b. Keeping the bank account of the user updated as soon as changes are encountered so that the data is in consistent state.

c. Blocking or seizing of the account of user on discovery of any illegal transaction.

d. Unblocking of ATM card that got blocked due to more than three unsuccessful login attempt.

e. Blocking of a lost/stolen ATM card on complaint of the user, only if he presents his verification and a FIR filed for that case.

f. Costantly monitor all the ATMs in the city to check whether any one of them is encountering any fault.

g. Immediately correct any fault discovered in any of the ATM.
h. Maintain the backup of all the accounts for reliability purposes.
i. Rollback all the changes made in an account during whose transaction an ATM got abnormal shutdown.

In case of loss of the ATM card. The user has to lodge a First Investigation Report(FIR) and present its one copy to bank officials for card blocking purposes.

A log of the following annexures is generated by the system:

User bank account details.
Updations made in the user account along with date, time and the changes made.

Schedule of fixed assets.

6 Other Requirements

None.

Appendix A: Glossary

AIMS - ATM Information Management System.

ATM - An unattended electronic machine in a public place, connected to a data system and related equipment and activated by a bank customer to obtain cash withdrawals and other banking services

Braille - A system of writing and printing for blind or visually impaired people, in which varied arrangements of raised dots representing letters and numerals are identified by touch.

CDMA - Code Division Multiple Access, a reliable data communication protocol.

CMS - Card Management Software developed by KPM Bank.

Dial-Up - A message format for low cost communications.

POS

Internet - An interconnected system of networks that connects computers around the world via the TCP/IP protocol.

Smart Card - Card without hardware which stores the user's private keys within a tamper proof software guard.

Tactile Keyboard - Special keyboard designed to aid the visually impaired.

TCP/IP - Transmission Control Protocol/Internet Protocol.