GAYA COLLEGE OF ENGINEERING, GAYA

SOFTWARE ENGINEERING

LAB MANUAL

(PCC CS 051 x 14)

PRABHAT KUMAR CHANDRA

ASSISTANT PROFESSOR

DEPARTMENT OF COMPUTER SC. & ENGINEERING

INDEX OF THE CONTENTS

Introduction to Software Engineering Lab

Hardware Requirements:

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

Software Requirements:

StarUML software/ Rational Rose, Windows XP/2000,ms-office

About Lab:

This lab deals with the analysis and design of a software problem .the tool used in a lab is rational rose .this tool is used for a object oriented design of a problem . We draw a uml diagram in a rational rose which deals with the objects and classes in a system .The **Unified Modeling Language** or **UML** is is a mostly graphical modelling language that is used to express designs. It is a standardized language in which to specify the artefacts and components of a software system. It is important to understand that the UML describes a notation and not a process. It does not put forth a single method or process of design, but rather is a standardized tool that can be used in a design process.

The Unified Modeling Language (UML) is a standard language for specifying, visualizing, constructing, and documenting the artifacts of software systems, as well as for business modeling and other non-software systems. The UML represents a collection of best engineering practices that have proven successful in the modeling of large and complex systems.[1] The UML is a very important part of developing object oriented software and the software development process. The UML uses mostly graphical notations to express the design of software projects. Using the UML helps project teams communicate, explore potential designs, and validate the architectural design of the software.

Goals of UML

The primary goals in the design of the UML were:

1. Provide users with a ready-to-use, expressive visual modeling language so they can develop and exchange meaningful models.
2. Provide extensibility and specialization mechanisms to extend the core concepts.
3. Be independent of particular programming languages and development processes.
4. Provide a formal basis for understanding the modeling language.
5. Encourage the growth of the OO tools market.
6. Support higher-level development concepts such as collaborations, frameworks, patterns and components.
7. Integrate best practices.

Why Use UML?

As the strategic value of software increases for many companies, the industry looks for techniques to automate the production of software and to improve quality and reduce cost and time-to-market. These techniques include component technology, visual programming, patterns and frameworks. Businesses also seek techniques to manage the complexity of systems as they increase in scope and scale. In particular, they recognize the need to solve recurring architectural problems, such as physical distribution, concurrency, replication, security, load balancing and fault tolerance.

List of Experiments:

1. Write down the problem statement for a suggested system of relevance.

2. Do requirement analysis and develop Software Requirement Specification Sheet (SRS) for suggested system.

3. To perform the function oriented diagram: Data Flow Diagram (DFD) and Structured chart.
4. To perform the user‘s view analysis for the suggested system: Use case diagram.
5. To draw the structural view diagram for the system: Class diagram, object diagram.
6. To draw the behavioral view diagram : State-chart diagram, Activity diagram

7. To perform the behavioral view diagram for the suggested system : Sequence diagram, Collaboration diagram

8. To perform the implementation view diagram: Component diagram for the system.

9. To perform the environmental view diagram: Deployment diagram for the system.

10. To perform various testing using the testing tool unit testing, integration testing for a sample code of the suggested system.

11. Perform Estimation of effort using FP Estimation for chosen system.

12. To Prepare time line chart/Gantt Chart/PERT Chart for selected software project.

Text Books:

1. K.K. Aggarwal & Yogesh Singh, ―Software Engineering‖, New Age International, 2005

2.Pankaj Jalote, ―An Integrated Approach to Software Engineering‖, Second Edition, Springer

Project List

(Student can opt their own project or anyone from below)

Choose any one project and do the following exercises for that project

a. Student Result Management System
b. Library management system
c. Inventory control system
d. Accounting system
e. Fast food billing system
f. Bank loan system
g. Blood bank system
h. Railway reservation system
i. Automatic teller machine
j. Video library management system
k. Hotel management system
l. Hostel management system
m. E-ticking
n. Share online trading
o. Hostel management system
p. Resource management system
q. Court case management system

1. Write the complete problem statement
2. Write the software requirement specification document
3. Draw the entity relationship diagram
4. Draw the data flow diagrams at level 0 and level 1
5. Draw use case diagram
6. Draw activity diagram of all use cases.
7. Draw state chart diagram of all use cases
8. Draw sequence diagram of all use cases
9. Draw collaboration diagram of all use cases
10. Assign objects in sequence diagram to classes and make class diagram .

Experiment. 1

AIM: To prepare PROBLEM STATEMENT for any project.

REQUIREMENTS:

Hardware Interfaces

- Pentium(R) 4 CPU 2.26 GHz, 128 MB RAM
- Screen resolution of at least 800 x 600 required for proper and complete viewing of screens. Higher resolution would not be a problem.
- CD ROM Driver

Software Interfaces

- Any window/ Linux operating system
- WordPad or Microsoft Word

THEORY:

The problem statement is the initial starting point for a project. It is basically a one to three page statement that everyone on the project agrees with that describes what will be done at a high level. The problem statement is intended for a broad audience and should be written in non-technical terms. It helps the non-technical and technical personnel communicate by providing a description of a problem. It doesn't describe the solution to the problem.

The input to requirement engineering is the problem statement prepared by customer.
It may give an overview of the existing system along with broad expectations from the new system.

The first phase of requirements engineering begins with requirements elicitation i.e. gathering of information about requirements. Here, requirements are identified with the help of customer and existing system processes. So from here begins the preparation of problem statement.
So, basically a problem statement describes **what** needs to be done without describing **how**.

Conclusion: The problem statement was written successfully by following the steps described above.

Experiment. 2

Aim: Understanding an SRS.

Requirements:

Hardware Requirements:

- PC with 300 megahertz or higher processor clock speed recommended; 233 MHz minimum required.
- 128 megabytes (MB) of RAM or higher recommended (64 MB minimum supported)
- 1.5 gigabytes (GB) of available hard disk space
- CD ROM or DVD Drive
- Keyboard and Mouse(compatible pointing device).

Software Requirements:

Rational Rose, Windows XP,

Theory:

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:

a) **Functionality**. What is the software supposed to do?
b) **External interfaces**. How does the software interact with people, the system‘s hardware, other hardware, and other software?
c) **Performance**. What is the speed, availability, response time, recovery time of various software functions, etc.?
d) **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.?

Chracteristics of a 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?

IEEE format of SRS documents

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 A
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

Sample of Software Requirement Specification document

ATM

AN AUTOMATED TELLER MACHINE

1. Introduction

The software **ATMExcl 3.0**^{TM} 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**^{TM} ,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**^{TM}. 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**^{TM}. 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:-

i. www.google.co.in

ii. www.winkipedia.com

iii. IEEE. Software Requirements Specification Std. 830-1993.

iv. Chevy Chase Bank, UMBC Branch.

v. 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 **ATMExcl 3.0**$^{\mathbf{TM}}$ 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, i.e. he can select whether the account is current account or savings account etc.
    - 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.

- Money 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.
- Balance Transfer:- Balance transfer shall be facilitated between any two accounts linked to the card for example saving and checking account.
- Balance Enquiry:- Balance enquiry for any account linked to the card shall be facilitated.

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.

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/-

- ✓ Before 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 user 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.
- Another constraint relating to the operating environment is that we are specific to Oracle Database.
- 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.
- At this stage no quantitive 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. External Interface Requirements

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 be of 10" VGA color type.
- 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.

3.1.3 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.

3.1.4 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.

5. Other 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 is 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:

1. Consistency – All code will be consistent with respect to the style. (This is implied when adhering to the standard).
2. Test cases – All functionality will be thoroughly tested

5.2 Software System 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

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.

Experiment No:3

AIM :-

To draw a sample ENTITY RELATIONSHIP DIAGRAM diagram for real project or system.

Hardware Requirements:

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

Software Requirements:

Rational Rose, Windows XP,

THEORY

Entity Relationship Diagrams are a major data modelling tool and will help organize the data in your project into entities and define the relationships between the entities. This process has proved to enable the analyst to produce a good database structure so that the data can be stored and retrieved in a most efficient manner.

Entity

A data entity is anything real or abstract about which we want to store data. Entity types fall into five classes: roles, events, locations, tangible things or concepts. E.g. employee, payment, campus, book. Specific examples of an entity are called **instances.** E.g. the employee John Jones, Mary Smith's payment, etc.

Relationship

A data relationship is a natural association that exists between one or more entities. E.g. Employees process payments. **Cardinality** defines the number of occurrences of one entity for a single occurrence of the related entity. E.g. an employee may process many payments but might not process any payments depending on the nature of her job.

Attribute

A data attribute is a characteristic common to all or most instances of a particular entity. Synonyms include property, data element, field. E.g. Name, address, Employee Number, pay rate are all attributes of the entity employee. An attribute or combination of attributes that uniquely identifies one and only one instance of an entity is called a **primary key** or **identifier**. E.g. Employee Number is a primary key for Employee.

AN ENTITY RELATIONSHIP DIAGRAM METHODOLOGY: (One way of doing it)

1. Identify Entities	Identify the roles, events, locations, tangible things or concepts about which the end-users want to store data.
2. Find Relationships	Find the natural associations between pairs of entities using a relationship matrix.
3. Draw Rough ERD	Put entities in rectangles and relationships on line segments connecting the entities.
4. Fill in Cardinality	Determine the number of occurrences of one entity for a single occurrence of the related entity.
5. Define Primary Keys	Identify the data attribute(s) that uniquely identify one and only one occurrence of each entity.
6. Draw Key-Based ERD	Eliminate Many-to-Many relationships and include primary and foreign keys in each entity.
7. Identify Attributes	Name the information details (fields) which are essential to the system under development.
8. Map Attributes	For each attribute, match it with exactly one entity that it describes.
9. Draw fully attributed ERD	Adjust the ERD from step 6 to account for entities or relationships discovered in step 8.
10. Check Results	Does the final Entity Relationship Diagram accurately depict the system data?

A SIMPLE EXAMPLE

A company has several departments. Each department has a supervisor and at least one employee. Employees must be assigned to at least one, but possibly more departments. At least one employee is assigned to a project, but an employee may be on vacation and not assigned to

any projects. The important data fields are the names of the departments, projects, supervisors and employees, as well as the supervisor and employee number and a unique project number.

1. Identify Entities

The entities in this system are Department, Employee, Supervisor and Project. One is tempted to make Company an entity, but it is a false entity because it has only one instance in this problem. True entities must have more than one instance.

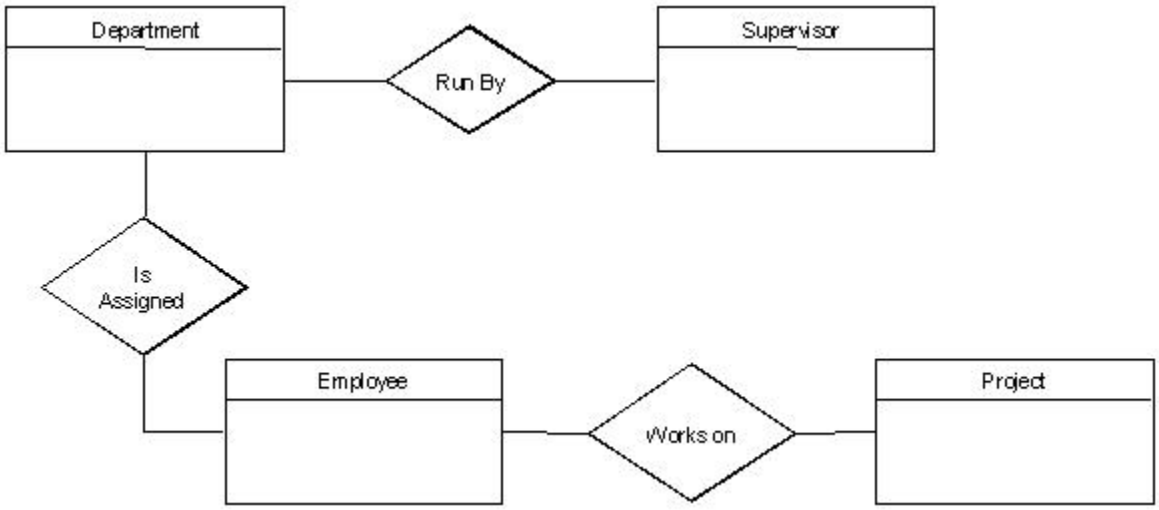
2. Find Relationships

We construct the following Entity Relationship Matrix:

	Department	**Employee**	**Supervisor**	**Project**
Department		is assigned	run by	
Employee	belongs to			works on
Supervisor	runs			
Project		uses		

3. Draw Rough ERD

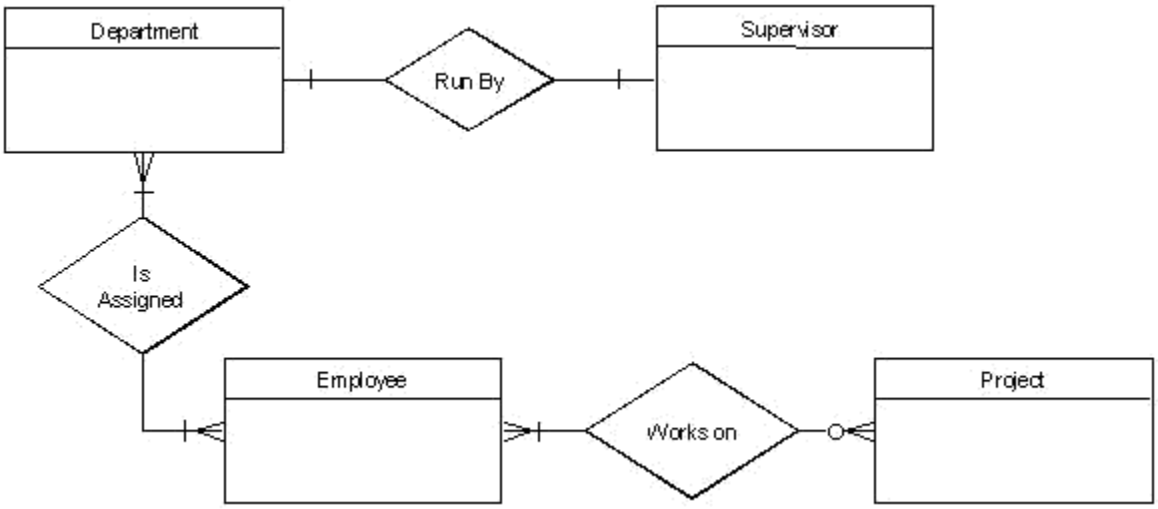
We connect the entities whenever a relationship is shown in the entity Relationship Matrix.



4. Fill in Cardinality

From the description of the problem we see that:

- Each department has exactly one supervisor.
- A supervisor is in charge of one and only one department.
- Each department is assigned at least one employee.
- Each employee works for at least one department.
- Each project has at least one employee working on it.
- An employee is assigned to 0 or more projects.



5. Define Primary Keys

The primary keys are Department Name, Supervisor Number, Employee Number, Project Number.

6. Draw Key-Based ERD

There are two many-to-many relationships in the rough ERD above, between Department and Employee and between Employee and Project. Thus we need the associative entities Department-Employee and Employee-Project. The primary key for Department-Employee is the concatenated key Department Name and Employee Number. The primary key for Employee-Project is the concatenated key Employee Number and Project Number.

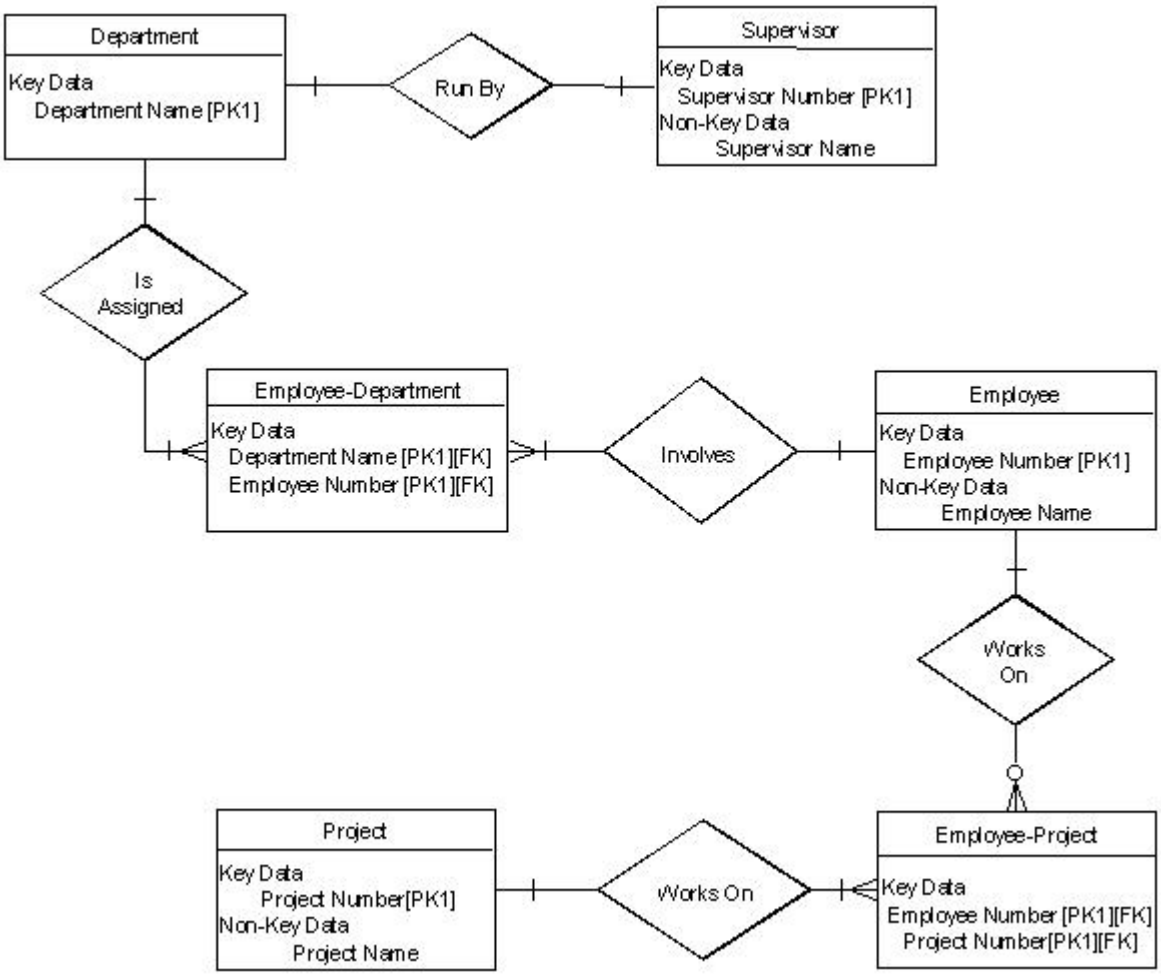
7. Identify Attributes

The only attributes indicated are the names of the departments, projects, supervisors and employees, as well as the supervisor and employee NUMBER and a unique project number.

8. Map Attributes

Attribute	**Entity**	**Attribute**	**Entity**
Department Name	Department	Supervisor Number	Supervisor
Employee Number	Employee	Supervisor Name	Supervisor
Employee Name	Employee	Project Name	Project
		Project Number	Project

9. Draw Fully Attributed ERD



10. Check Results

The final ERD appears to model the data in this system well.

FURTHER DISCUSSION:

Step 1. Identify Entities

A data entity is anything real or abstract about which we want to store data. Entity types fall into five classes: roles, events, locations, tangible things, or concepts. The best way to identify entities is to ask the system owners and users to identify things about which they would like to capture, store and produce information. Another source for identifying entities is to study the forms, files, and reports generated by the current system. E.g. a student registration form would refer to Student (a role), but also Course (an event), Instructor (a role), Advisor (a role), Room (a location), etc.

Step 2. Find Relationships

There are natural associations between pairs of entities. Listing the entities down the left column and across the top of a table, we can form a relationship matrix by filling in an active verb at the intersection of two entities which are related. Each row and column should have at least one relationship listed or else the entity associated with that row or column does not interact with the rest of the system. In this case, you should question whether it makes sense to include that entity in the system.

. A student is enrolled in one or more courses

subject verb objects

Step 3. Draw Rough ERD

Using rectangles for entities and lines for relationships, we can draw an Entity Relationship Diagram (ERD).

Step 4. Fill in Cardinality

At each end of each connector joining rectangles, we need to place a symbol indicating the minimum and maximum number of instances of the adjacent rectangle there are for one instance of the rectangle at the other end of the relationship line. The placement of these numbers is often confusing. The first symbol is either 0 to indicate that it is possible for no instances of the entity joining the connector to be related to a given instance of the entity on the other side of the relationship, 1 if at least one instance is necessary or it is omitted if more than one instance is required. For example, more than one student must be enrolled in a course for it to run, but it is possible for no students to have a particular instructor (if they are on leave).

The second symbol gives the maximum number of instances of the entity joining the connector for each instance of the entity on the other side of the relationship. If there is only one such instance, this symbol is 1. If more than 1, the symbol is a crows foot opening towards the rectangle.

If you read it like a sentence, the first entity is the subject, the relationship is the verb, the cardinality after the relationship tells how many direct objects (second entity) there are.

I.e. A student is enrolled in one or more courses

subject verb objects

Step 5. Define Primary Keys

For each entity we must find a unique primary key so that instances of that entity can be distinguished from one another. Often a single field or property is a primary key (e.g. a Student ID). Other times the identifier is a set of fields or attributes (e.g. a course needs a department identifier, a course number, and often a section number; a Room needs a Building Name and a Room Number). When the entity is written with all its attributes, the primary key is underlined.

Step 6. Draw Key-Based ERD

Looking at the Rough Draft ERD, we may see some relationships which are non-specific or many-to-many. I.e., there are crows feet on both ends of the relationship line. Such relationships spell trouble later when we try to implement the related entities as data stores or data files, since each record will need an indefinite number of fields to maintain the many-to-many relationship.

Fortunately, by introducing an extra entity, called an associative entity for each many-to-many relationship, we can solve this problem. The new associative entity's name will be the hyphenation of the names of the two originating entities. It will have a concatenated key consisting of the keys of these two entities. It will have a 1-1 relationship with each of its parent entities and each parent will have the same relationship with the associative entity that they had with each other before we introduced the associative entity. The original relationship between the parents will be deleted from the diagram.

The key-based ERD has no many-to-many relationships and each entity has its primary and foreign keys listed below the entity name in its rectangle.

Step 7. Identify Attributes

A data attribute is a characteristic common to all or most instances of a particular entity. In this step we try to identify and name all the attributes essential to the system we are studying without trying to match them to particular entities. The best way to do this is to study the forms, files and reports currently kept by the users of the system and circle each data item on the paper copy. Cross out those which will not be transferred to the new system, extraneous items such as signatures, and constant information which is the same for all instances of the form (e.g. your company name and address). The remaining circled items should represent the attributes you need. You should always verify these with your system users. (Sometimes forms or reports are out of date.)

Step 8. Map Attributes

For each attribute we need to match it with exactly one entity. Often it seems like an attribute should go with more than one entity (e.g. Name). In this case you need to add a modifier to the attribute name to make it unique (e.g. Customer Name, Employee Name, etc.) or determine which entity an attribute "best' describes. If you have attributes left over without corresponding entities, you may have missed an entity and its corresponding relationships. Identify these missed entities and add them to the relationship matrix now.

Step 9. Draw Fully-Attributed ERD

If you introduced new entities and attributes in step 8, you need to redraw the entity relationship diagram. When you do so, try to rearrange it so no lines cross by putting the entities with the

most relationships in the middle. If you use a tool like Systems Architect, redrawing the diagram is relatively easy.

Even if you have no new entities to add to the Key-Based ERD, you still need to add the attributes to the Non-Key Data section of each rectangle. Adding these attributes automatically puts them in the repository, so when we use the entity to design the new system, all its attributes will be available.

Step 10. Check Results

Look at your diagram from the point of view of a system owner or user. Is everything clear? Check through the Cardinality pairs. Also, look over the list of attributes associated with each entity to see if anything has been omitted.

Conclusion: The entity relationship diagram was made successfully by following the steps described above.

Experiment No:4

AIM: To prepare DATA FLOW DIAGRAM for any project.

REQUIREMENTS:

Hardware Interfaces

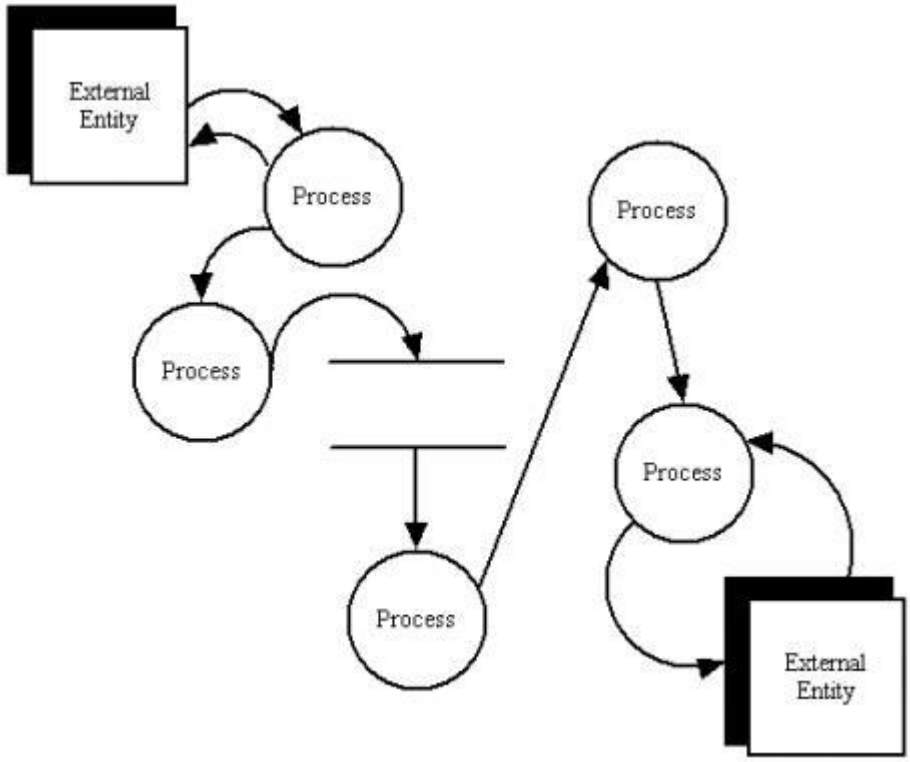
- Pentium(R) 4 CPU 2.26 GHz, 128 MB RAM
- Screen resolution of at least 800 x 600 required for proper and complete viewing of screens. Higher resolution would not be a problem.
- CD ROM Driver

Software Interfaces

- Any window-based operating system (Windows 95/98/2000/XP/NT)
- WordPad or Microsoft Word

THEORY

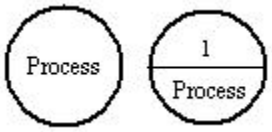
Data flow diagrams illustrate how data is processed by a system in terms of inputs and outputs.



Data Flow Diagram Notations

You can use two different types of notations on your data flow diagrams: Yourdon & Coad or Gane & Sarson.

Process Notations



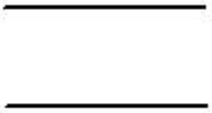
Yourdon and Coad
Process Notations

Gane and Sarson
Process Notation

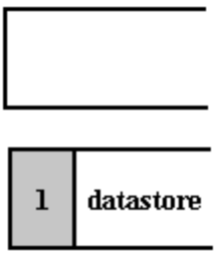
Process

A process transforms incoming data flow into outgoing data flow.

Datastore Notations



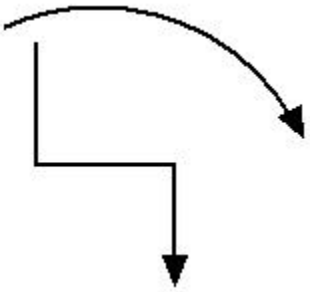
Yourdon and Coad
Datastore Notations



Gane and Sarson
Datastore Notations

DataStore
Datastores are repositories of data in the system. They are sometimes also referred to as files.

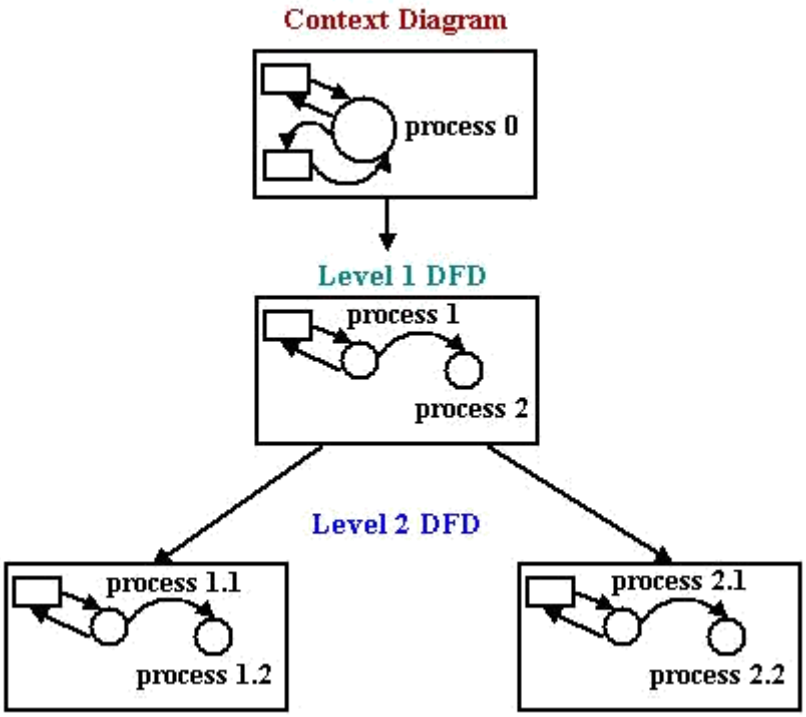
Dataflow Notations



Dataflow
Dataflows are pipelines through which packets of information flow. Label the arrows with the name of the data that moves through it.

HOW TO DRAW DATA FLOW DIAGRAMS (cont'd)

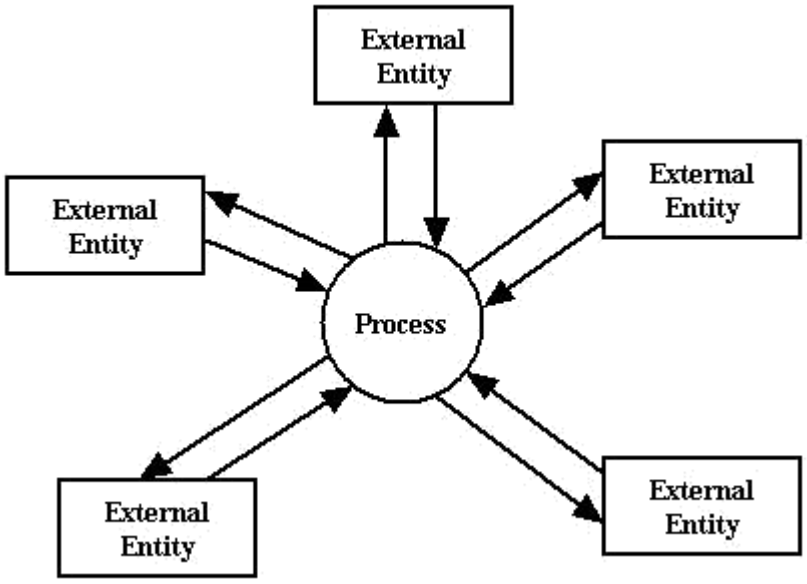
Data Flow Diagram Layers
Draw data flow diagrams in several nested layers. A single process node on a high level diagram can be expanded to show a more detailed data flow diagram. Draw the context diagram first, followed by various layers of data flow diagrams.



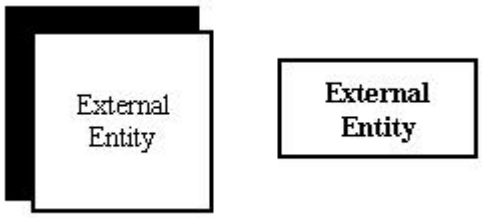
The nesting of data flow layers

Context Diagrams

A context diagram is a top level (also known as Level 0) data flow diagram. It only contains one process node (process 0) that generalizes the function of the entire system in relationship to external entities.



External Entity Notations

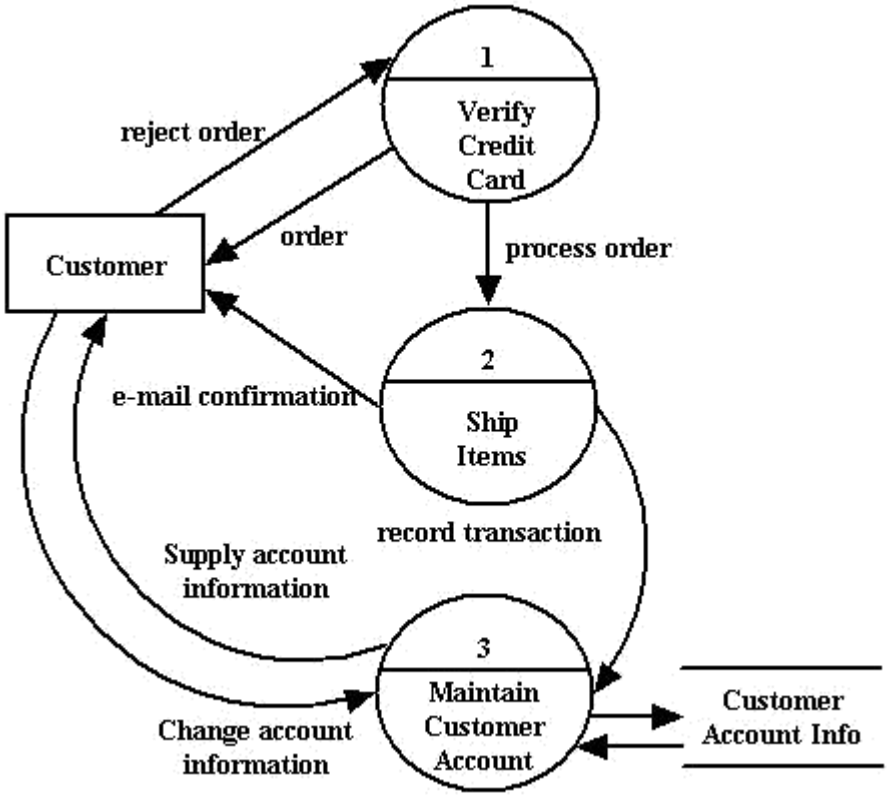


External Entity

External entities are objects outside the system, with which the system communicates. External entities are sources and destinations of the system's inputs and outputs.

DFD levels

The first level DFD shows the main processes within the system. Each of these processes can be broken into further processes until you reach pseudocode.



An example first-level data flow diagram

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

Experiment No. 5

Aim:

Steps to draw the Use Case Diagram using Rational Rose.

Hardware Requirements:

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

Software Requirements:

Rational Rose, Windows XP,

Theory:

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.

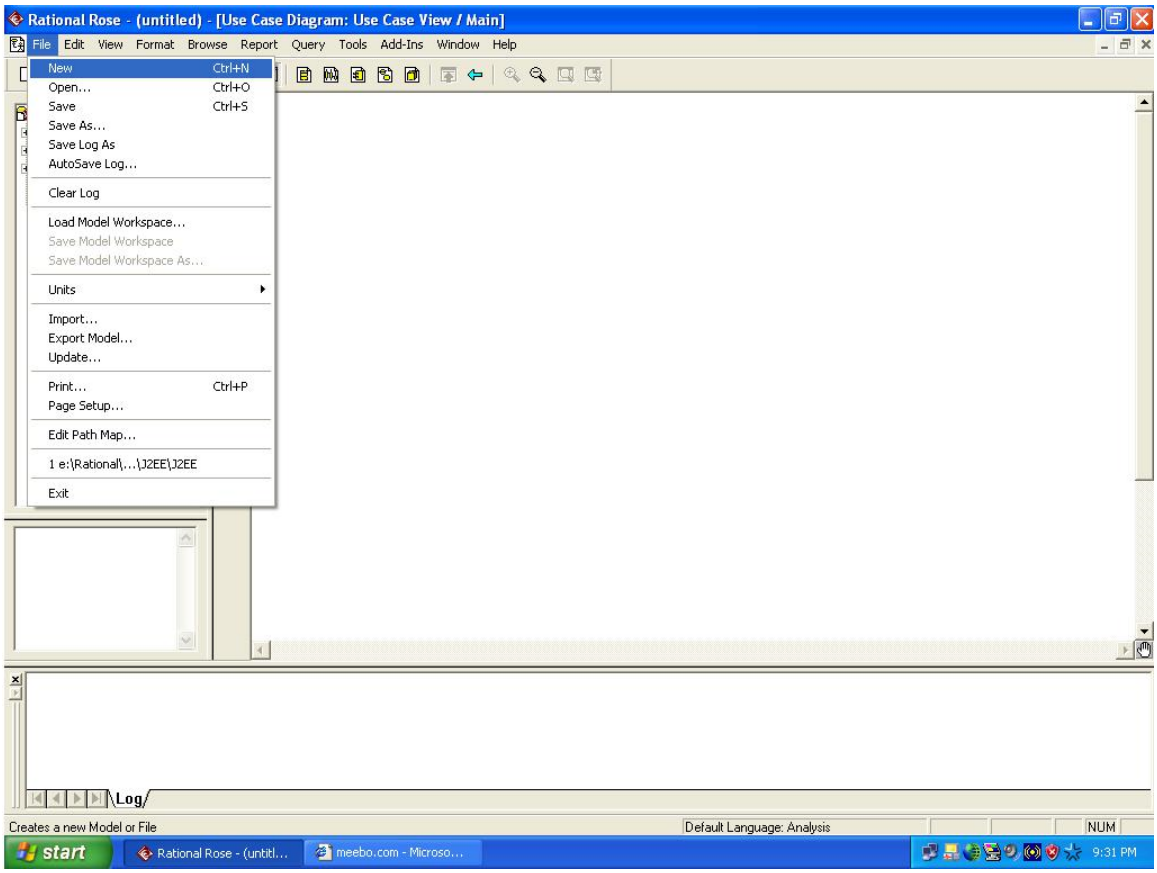
1. Indicate Release Scope with a System Boundary Box.
2. Avoid Meaningless System Boundary Boxes.

Creating Use Case Diagrams

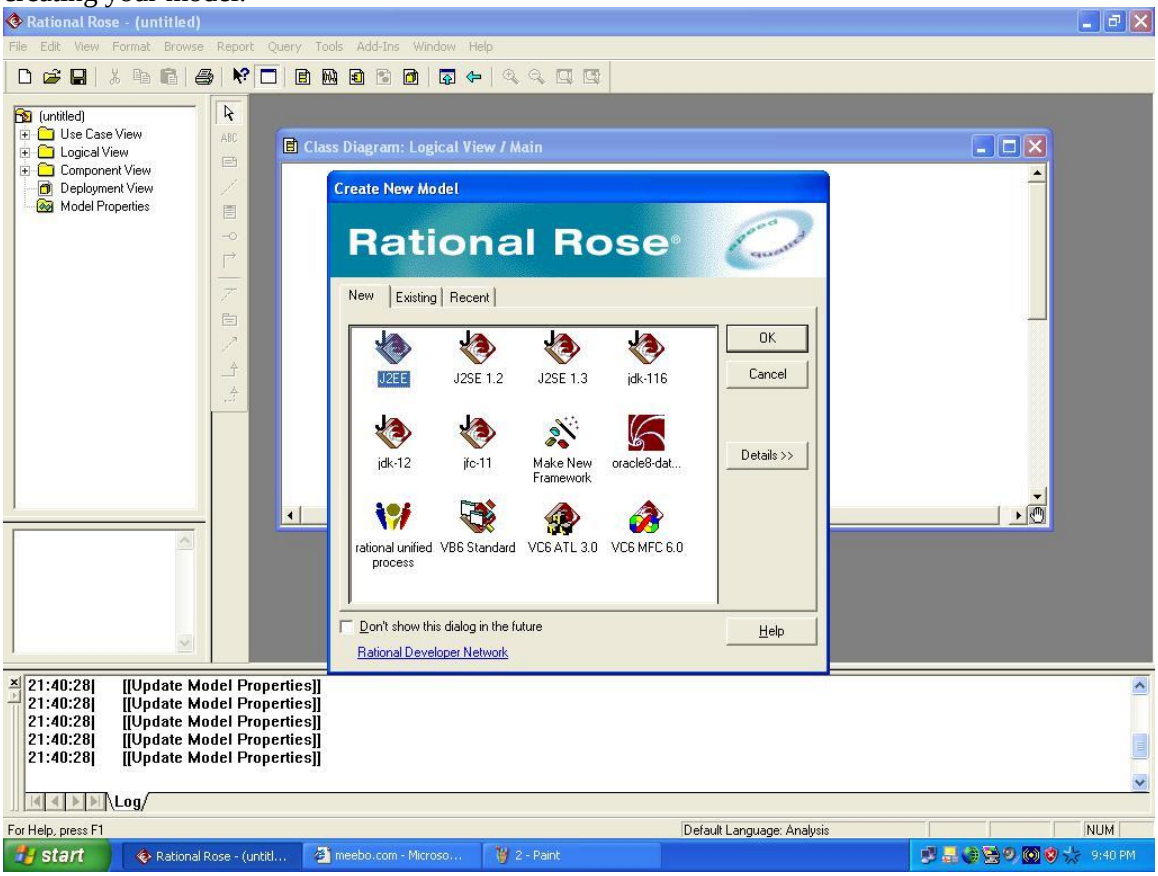
we start by identifying as many actors as possible. You should ask how the actors interact with the system to identify an initial set of use cases. Then, on the diagram, you connect the actors with the use cases with which they are involved. If actor supplies information, initiates the use case, or receives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

Procedure (for rational rose):

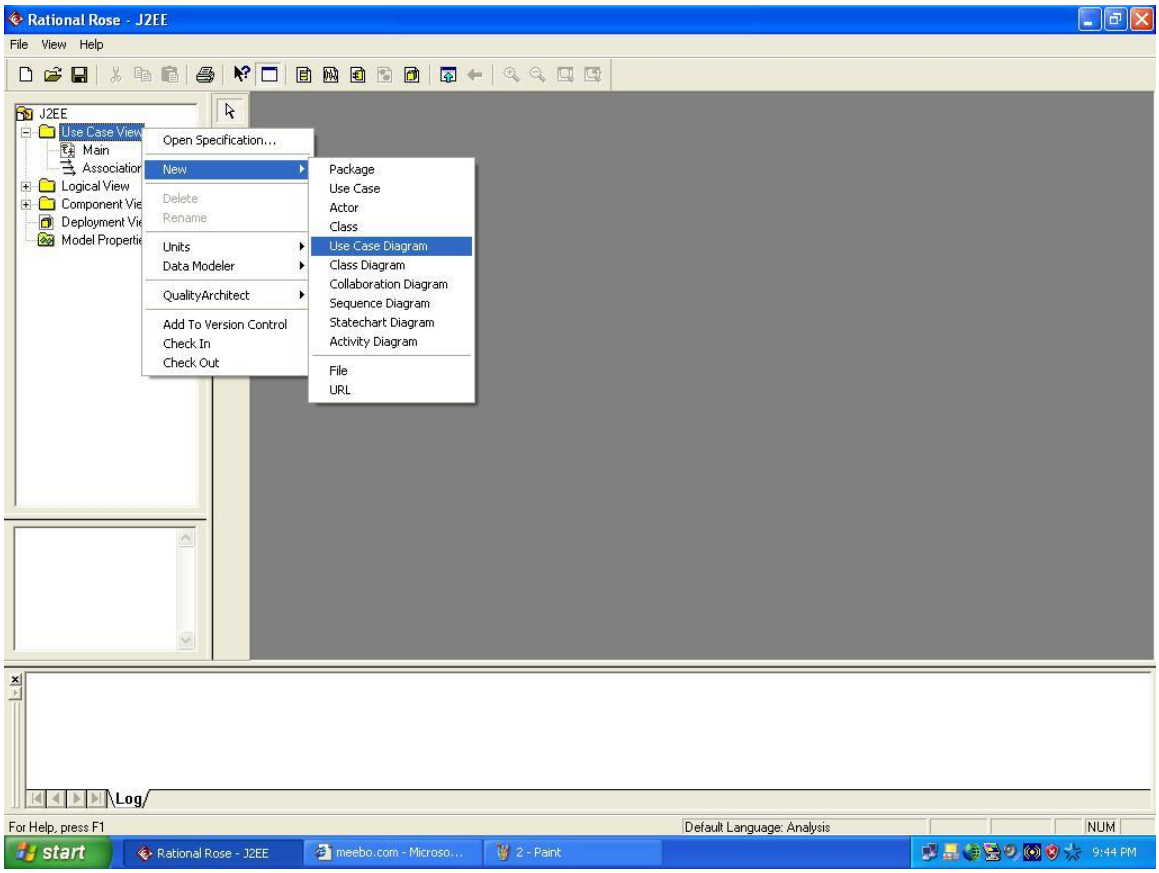
- Click on the File menu and select New.



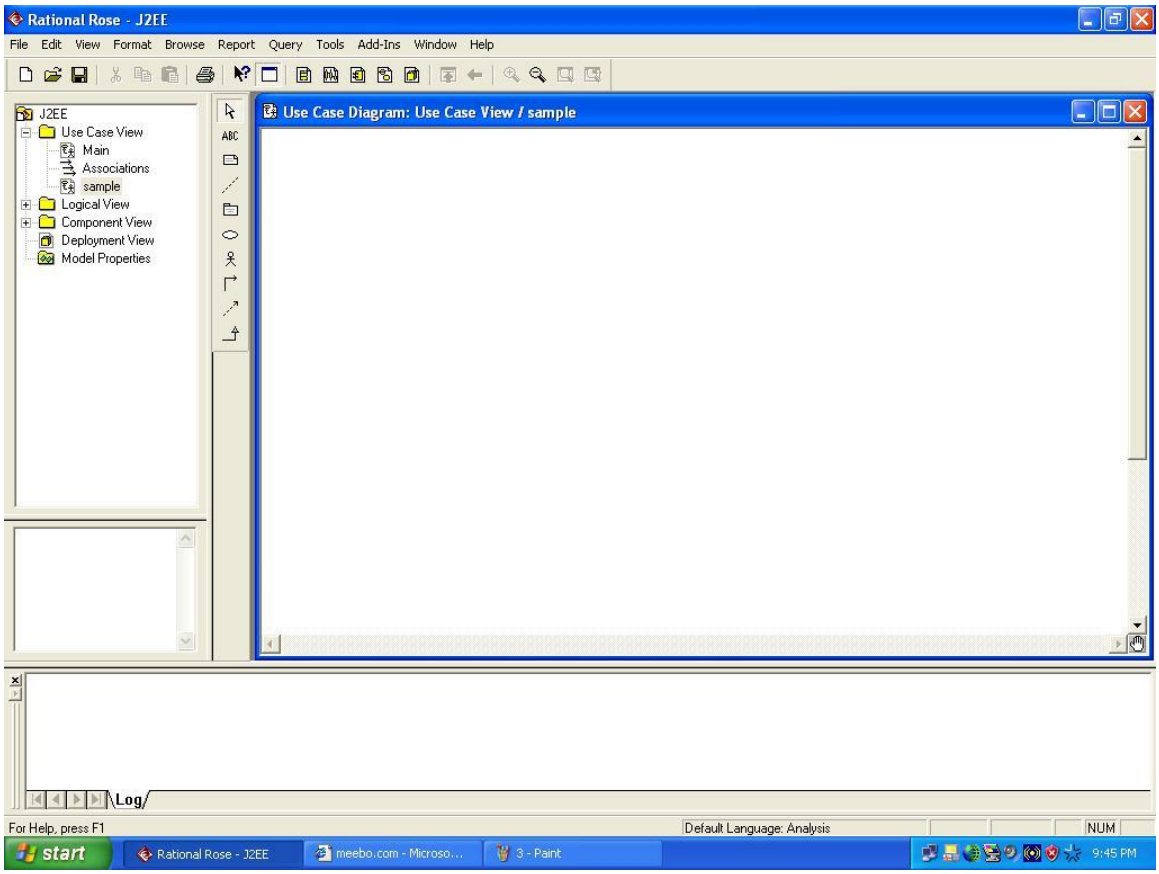
- Now from the Dialogue Box that appears ,select the language which you want to use for creating your model.



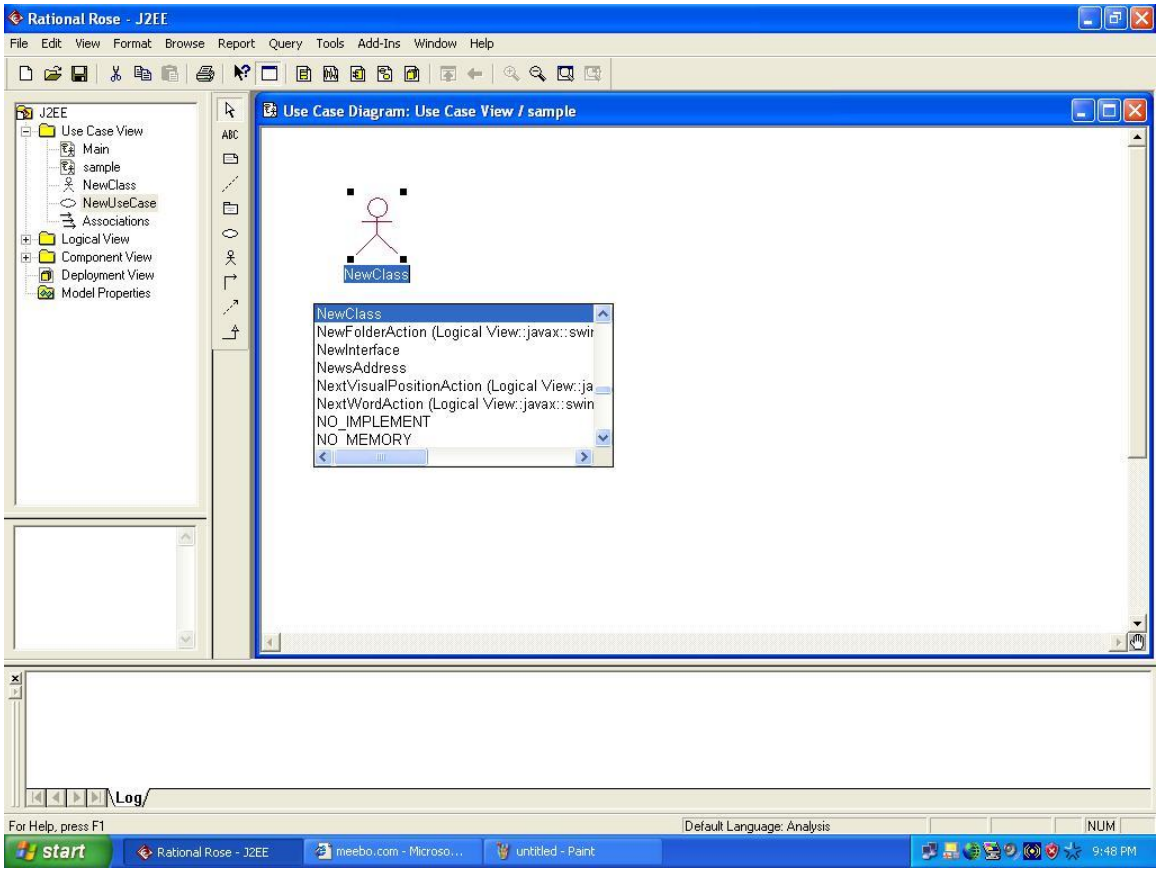
- In the left hand side window of Rational Rose right click on ―Use Case view‖ and select New>Use Case Diagram.



- Enter the name of new Use Case file in the space provided,and then click on that file name.

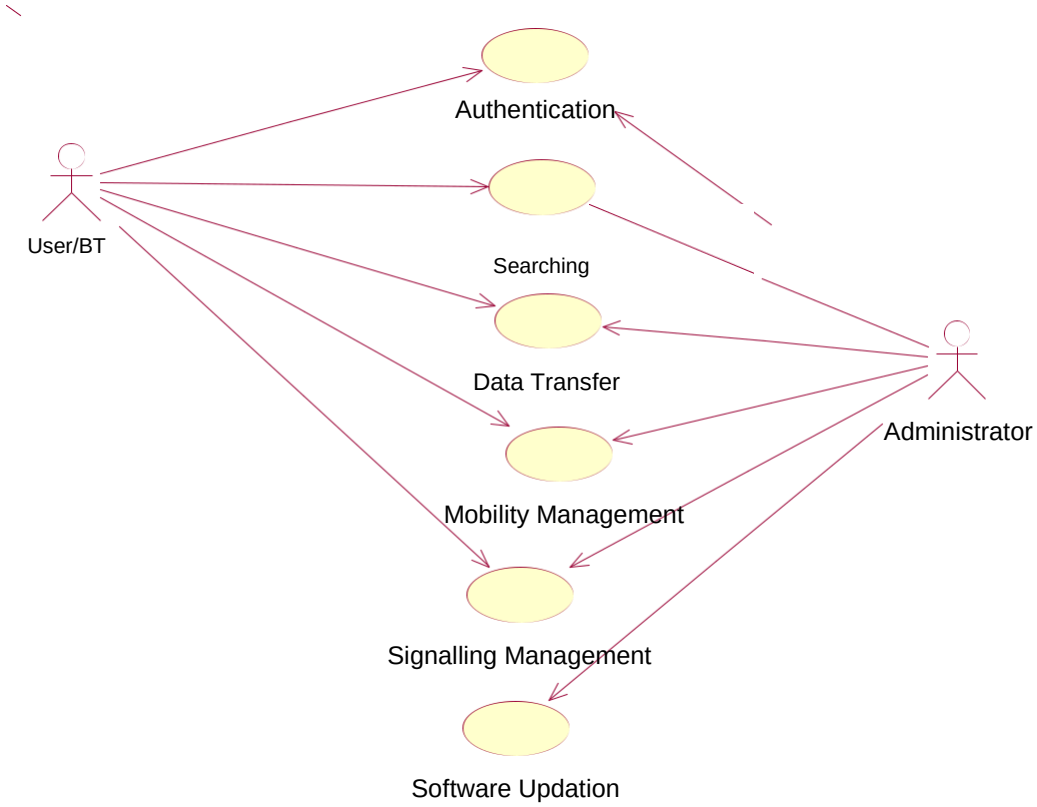


- You can now use the window that appears on right hand side to draw your Use Case diagram using the buttons provided on the vertical toolbar.

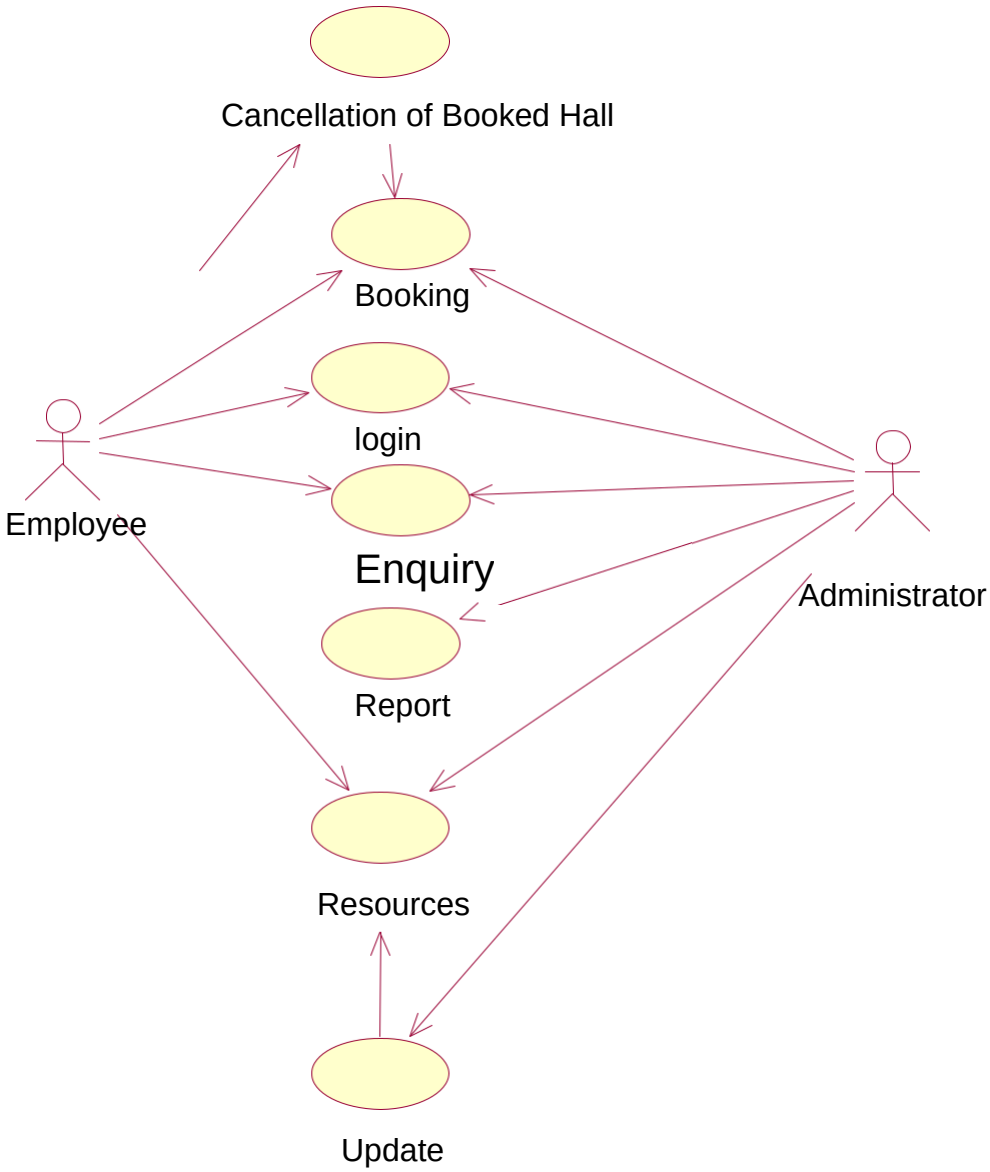


Conclusion: The use case diagram was made successfully by following the steps described above.

Some Sample Use Case Diagrams are given below for illustration purpose:



Use Case Diagram for Bluetooth Software



Use Case Diagram for Resource Management

Experiment No: 06

AIM :-

To draw a sample activity diagram for real project or system.

Hardware Requirements:

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

Software Requirements:

Rational Rose, Windows XP,

THEORY

UML 2 activity diagrams are typically used for business process modeling, for modeling the logic captured by a single use case 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

1.General Guidelines

2.Activities

3.Decision Points

4.Guards

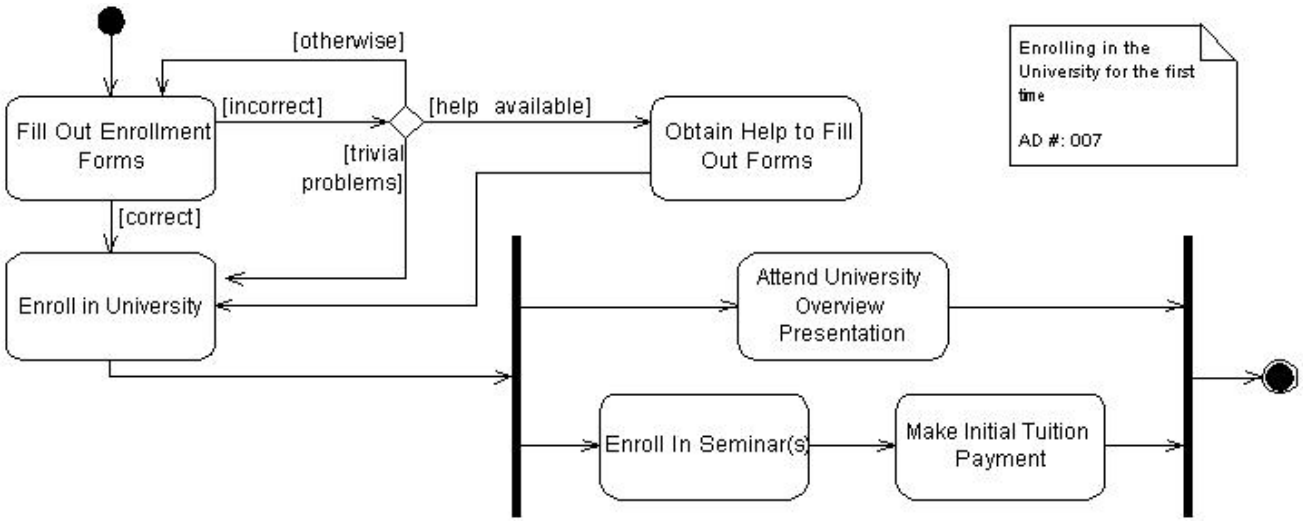
5.Parallel Activities

6.Swimlane Guidelines

7.Action-Object Guidelines

1. General Guidelines

Figure1. Modeling a business process with a UML Activity Diagram.



1. 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.
2. 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.
3. 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.

2. 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.

1. 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.

2. 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.

3. Decision Points

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

1. 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.
2. 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.

4. Guards

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

1. Each Transition Leaving a Decision Point Must Have a Guard
2. 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.
3. 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 0.
4. 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.
5. Apply a [Otherwise] Guard for ―Fall Through‖ Logic.
6. 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.

1. 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.
2. Forks Have One Entry Transition.
3. Joins Have One Exit Transition
4. 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.

6. 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).

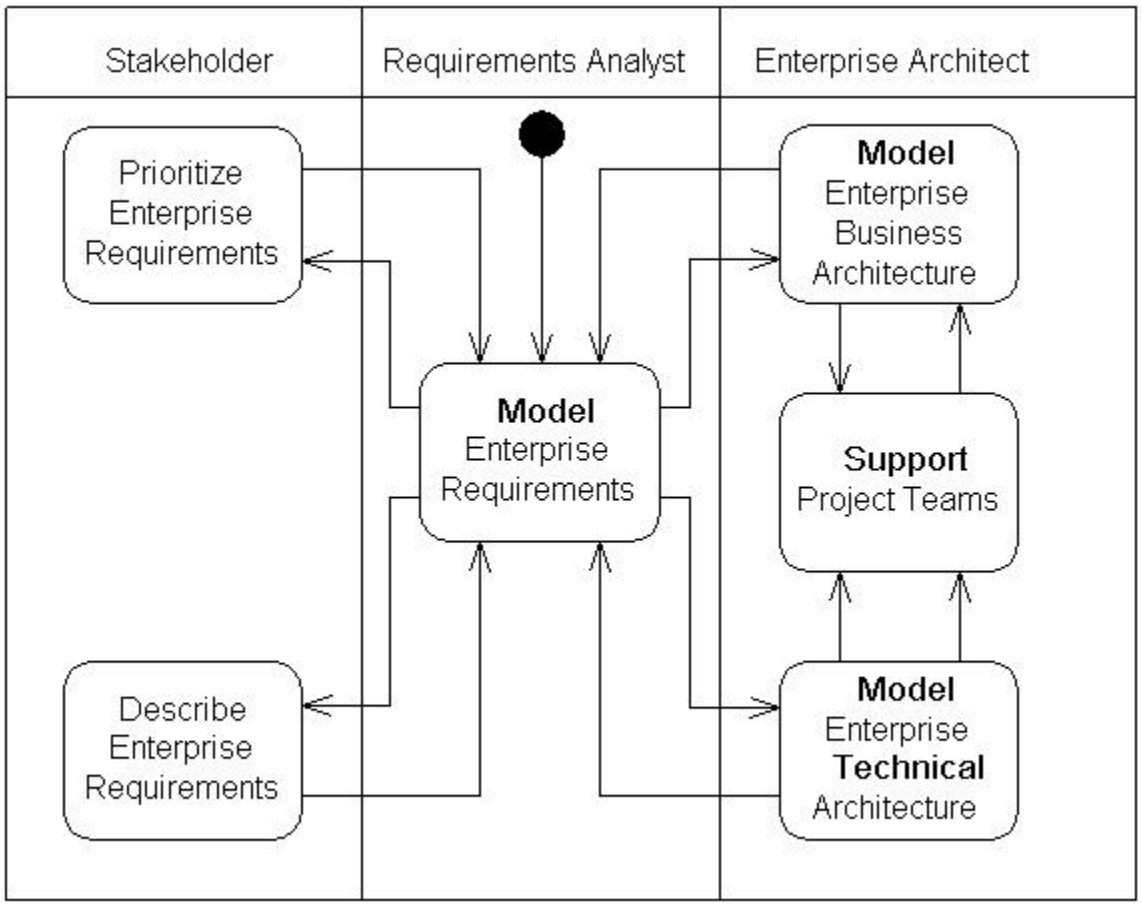
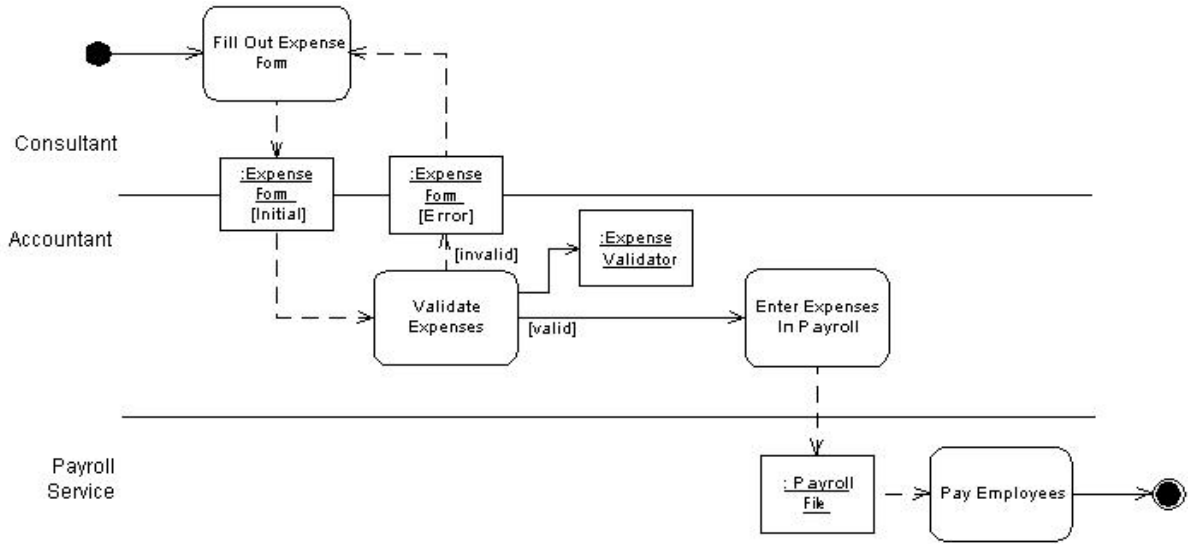


Figure 3. Submitting expenses.



1. Order Swimlanes in a Logical Manner.
2. 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.
3. Have Less Than Five Swimlanes.
4. Consider Swimareas For Complex Diagrams.
5. Swimareas Suggest The Need to Reorganize Into Smaller Activity Diagrams.
6. 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.

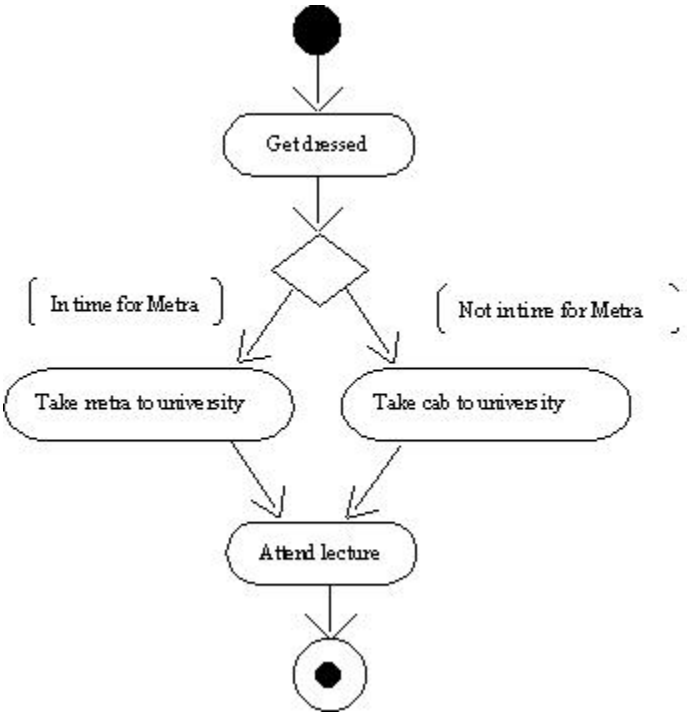
1. Place Shared Action Objects on Swimlane Separators
2. When An Object Appears Several Time Apply State Names
3. State Names Should Reflect the Lifecycle Stage of an Action Object
4. Show Only Critical Inputs and Outputs
5. Depict Action Objects As Smaller Than Activities

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

SAMPLE ACTIVITY DIAGRAMS:

SAMPLE 1:

Let us consider the example of attending a course lecture, at 8 am.



An example Activity diagram

As you can see in Figure , the first activity is to get dressed to leave for the lecture. A decision then has to be made, depending on the time available for the lecture to start, and the timings of the public trains (metra). If there is sufficient time to catch the train, then take the train; else, flag down a cab to the University. The final activity is to actually attend the lecture, after which the Activity diagram terminates.

SAMPLE 2:

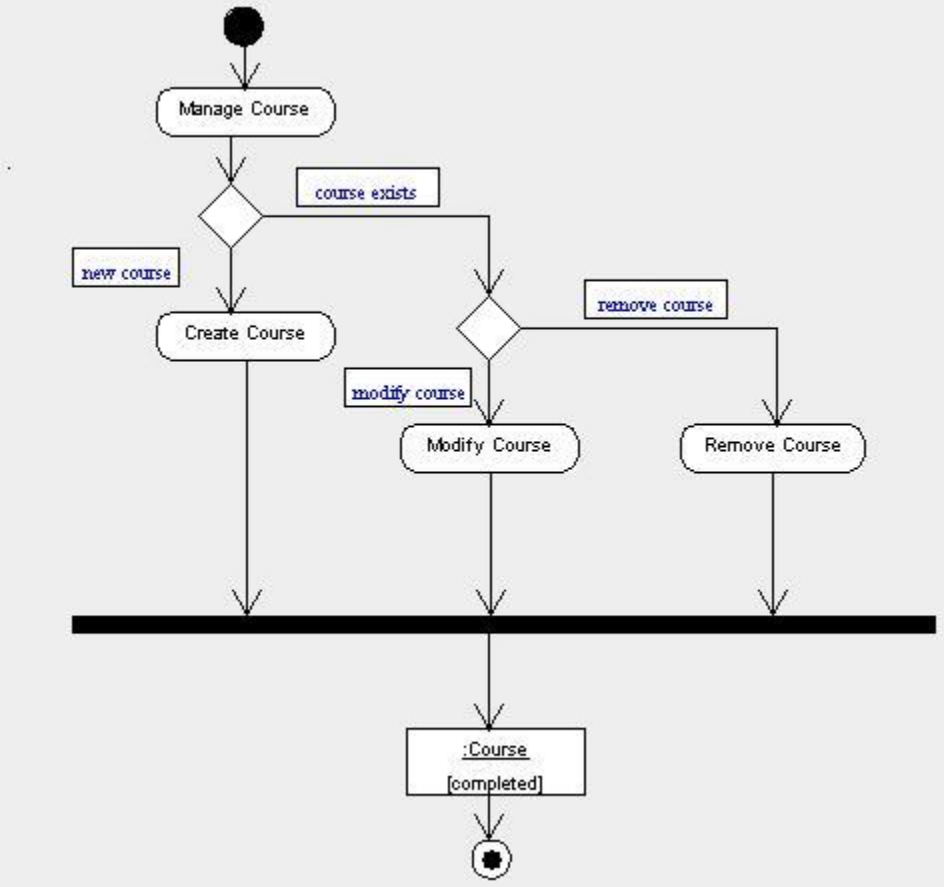
Identifying the activities and transitions for managing course information

The course administrator is responsible for managing course information in the Courseware Management System. As part of managing the course information, the course administrator carries out the following activities:

- Check if course exists
- If course is new, proceed to the "Create Course" step
- If course exists, check what operation is desired—whether to modify the course or remove the course
- If the modify course operation is selected by the course administrator, the "Modify Course" activity is performed
- If the remove course operation is selected by the course administrator, the "Remove Course" activity is performed

In the first step in this Activity diagram, the system determines whether the course that is to be managed is a new course or an existing course. For managing a new course, a separate activity, "Create Course," is performed. On the other hand, if a course exists, the course administrator can perform two different activities—modify an existing course or remove an existing course. Hence, the system checks the type of operation desired based on which two separate activities can be performed—"Modify Course" or "Remove Course".

Activity diagram



Experiment No. 7

AIM: To prepare STATE CHART DIAGRAM for any project.

REQUIREMENTS:

Hardware Interfaces

- Pentium(R) 4 CPU 2.26 GHz, 128 MB RAM
- Screen resolution of at least 800 x 600 required for proper and complete viewing of screens. Higher resolution would not be a problem.
- CD ROM Driver

Software Interfaces

- Any window-based operating system(Windows98/2000/XP/NT)
- IBM Rational Rose Software

THEORY:

- ✓ State Chart Diagrams provide a way to model the various states in which an object can exist.
- ✓ There are two special states: the start state and the stop state.

The **Start state** is represented by a **block dot**.

The **Stop state** is represented by a **bull's eye**.

- ✓ A condition enclosed in square brackets is called a **guard condition,** and controls when a transition can or cannot occur.
- ✓ **Process** that occur while an object is in certain state are called **actions.**

STEPS TO DRAW STATE CHART DIAGRAM IN RATIONAL ROSE SOFTWARE

- To insert new state diagram secondary click on Logical View.
  - ❖ **Select---New----State chart Diagram**.
  - ❖ A new diagram will be created, type in a name for the new diagram.
- Now double click on the new diagram to open it on the stage.
  - ❖ To begin the diagram click on the **"START STATE"** button.
  - ❖ Place a start state icon on the diagram by clicking the mouse once.
- Now add states to the diagram, these make up the content of the diagram. Click on the state button. Place the instances for each state into the diagram and type in names for them.
- Now arrange the states to fill the diagram better. Drag the states to new positions to make the easiest layout to work with.

- Add an end state to the diagram by clicking the **"END STATE"** button. Place an instance into the diagram. Now add relationships to the diagram.

- Click on the **"STATE TRANSITION"** button and drag arrows between the appropriate states.

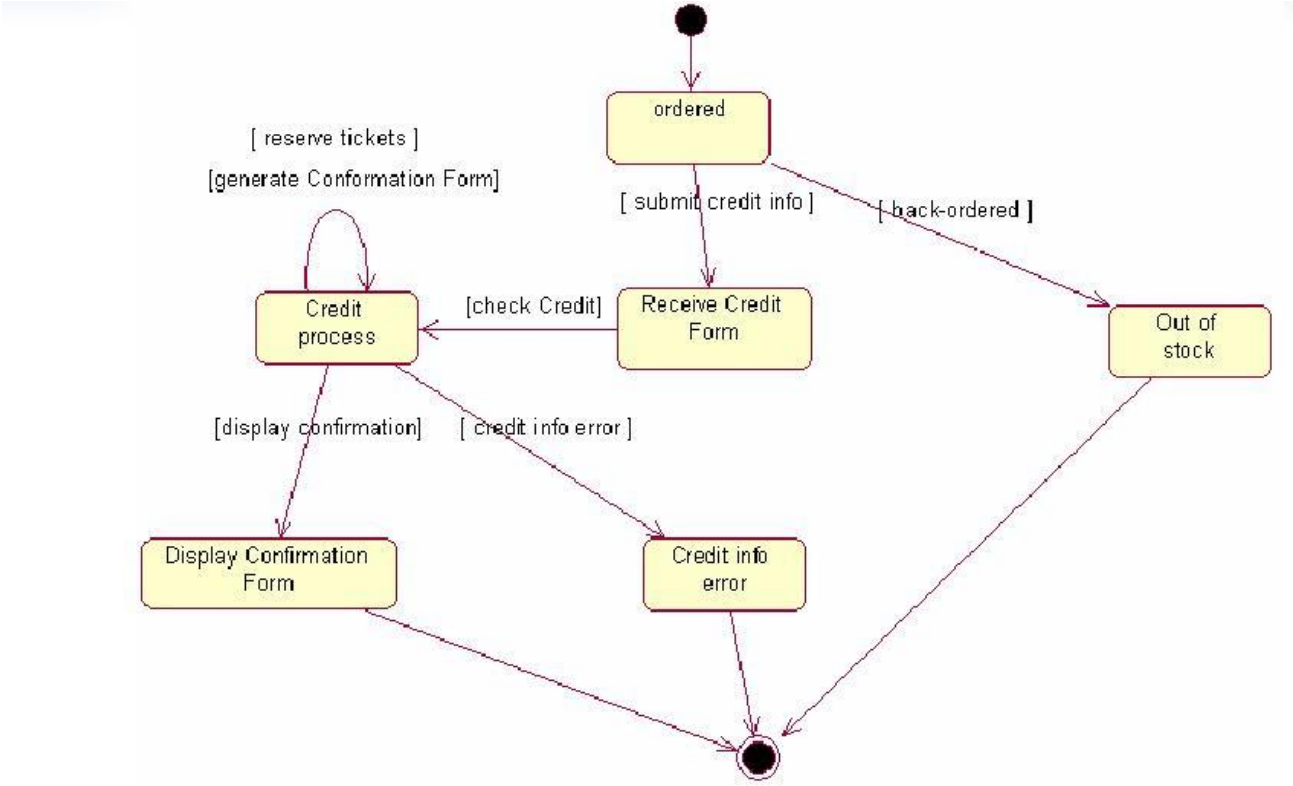
- To edit the specification secondary click on the relation lines and select **"OPEN SPECIFICATION"** button. Add a name for the event in the specification. Then click on ―apply‖ and then on ―OK‖ button.

- Add details to the specifications of the other relationships in the same way.

- There may be instances on the diagram where a state can join more than one state. In this case add a relationship in the same way. Then enter the specification for the new state.

Conclusion: The state chart diagram was made successfully by following the steps described above.

This is how a state chart diagram is made.



A STATE CHART DIAGRAM FOR RESERVATION OF TICKETS & DISPLAY OF CONFIRMATION FORM.

Experiment No: 8

Aim:

Steps to draw the Sequence Diagram using Rational Rose.

Hardware Requirements:

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

Software Requirements:

Rational Rose, 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.

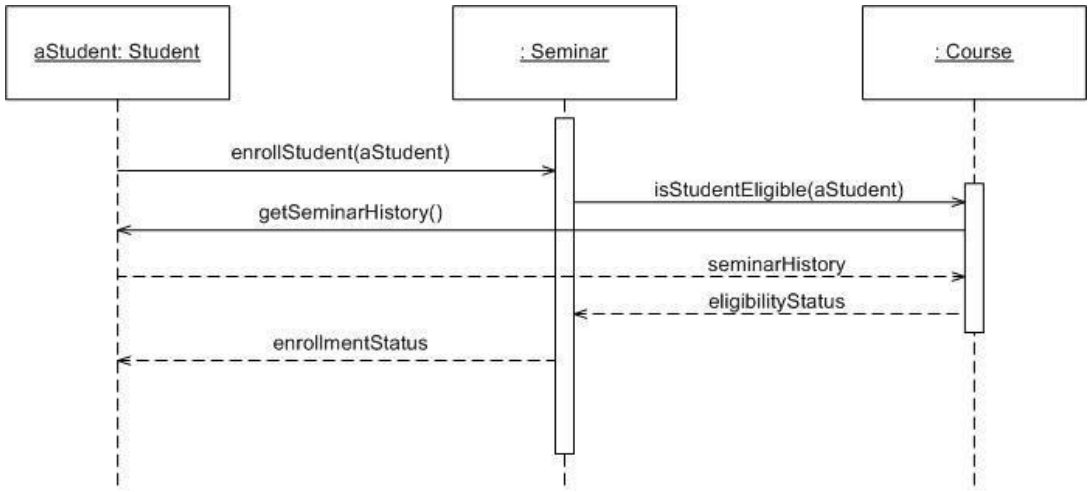
Sequence diagrams are typically used to model:

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.

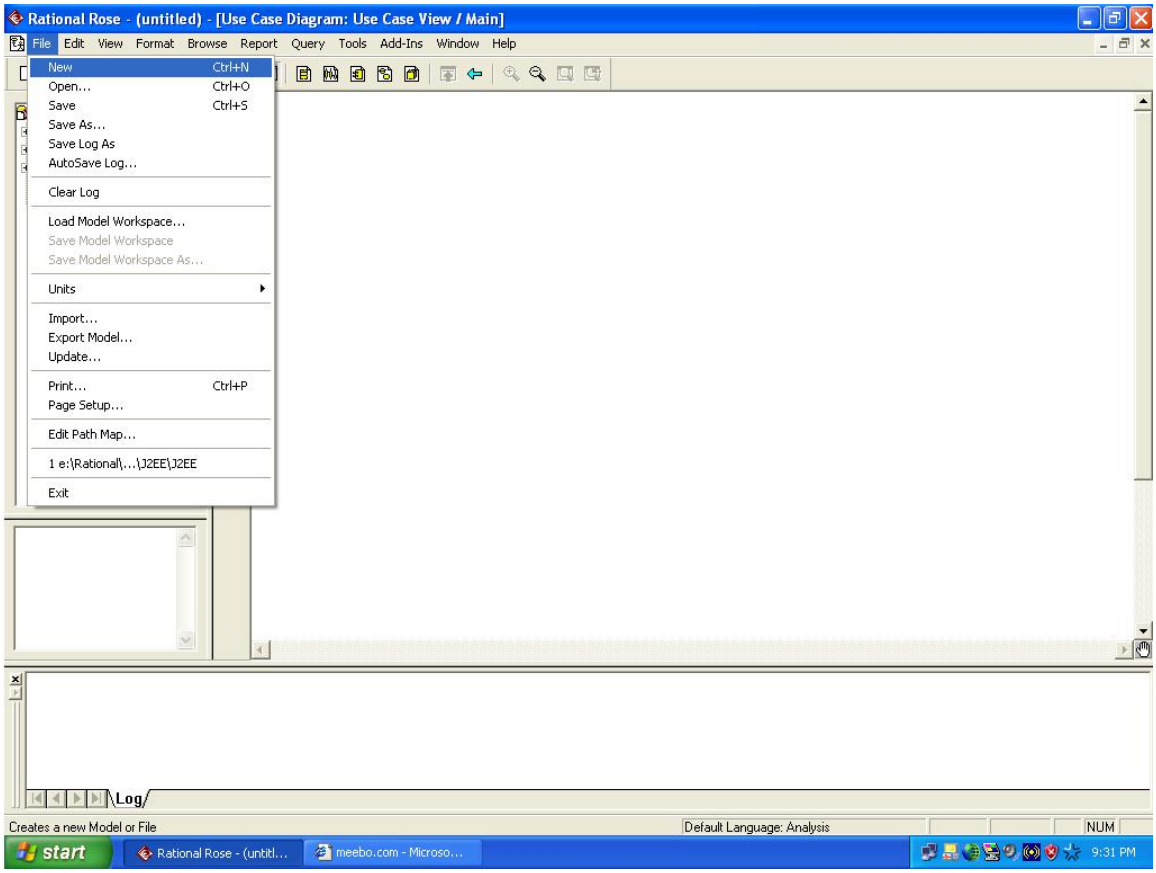
You should then work through the logic with at least one more person, laying out classifiers across the top as you need them. . The heart of the diagram is in the messages, which you add to the diagram one at a time as you work through the logic. You should rarely indicate return values, instead you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

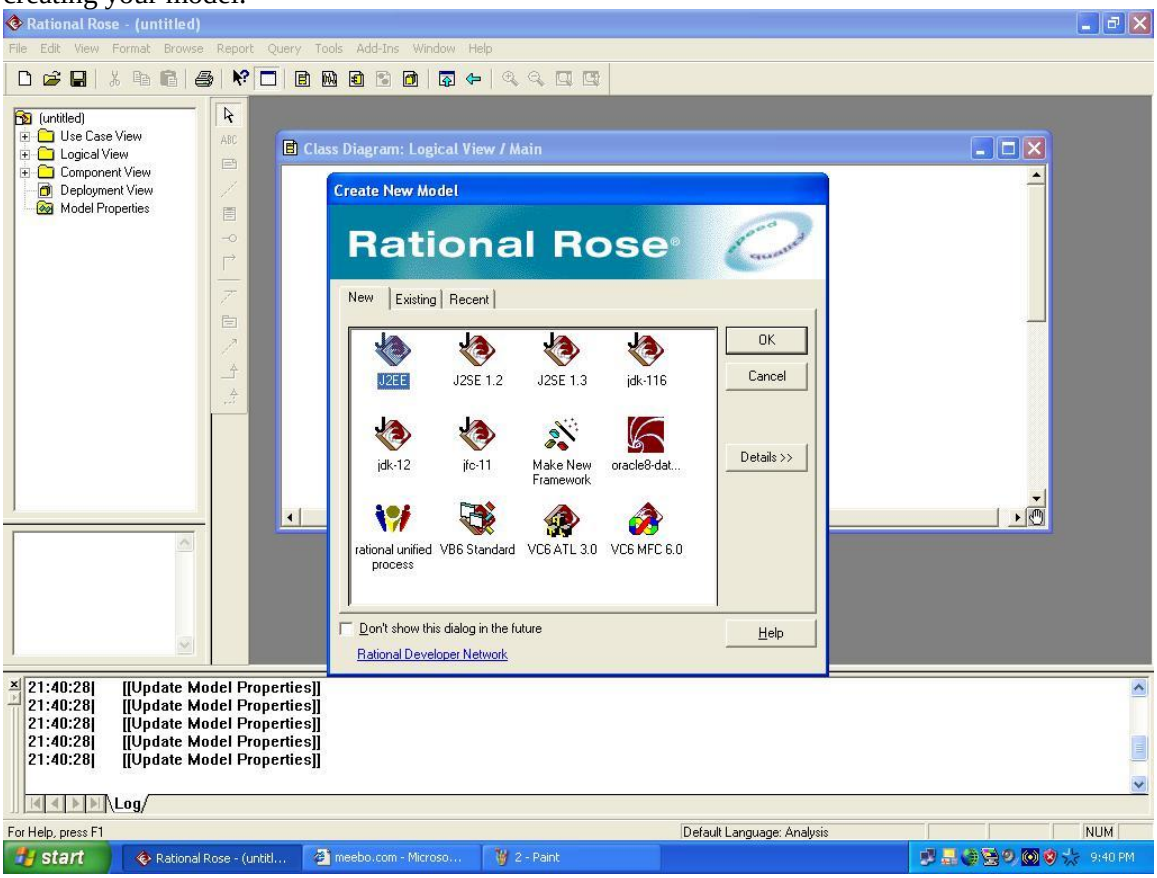
Regarding style issues for sequence diagramming, prefer drawing messages going from left-to-right and return values from right-to-left, although that doesn‘t always work with complex objects/classes. Justify the label on messages and return values, so they are closest to the arrowhead. Also prefer to 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 .

Procedure

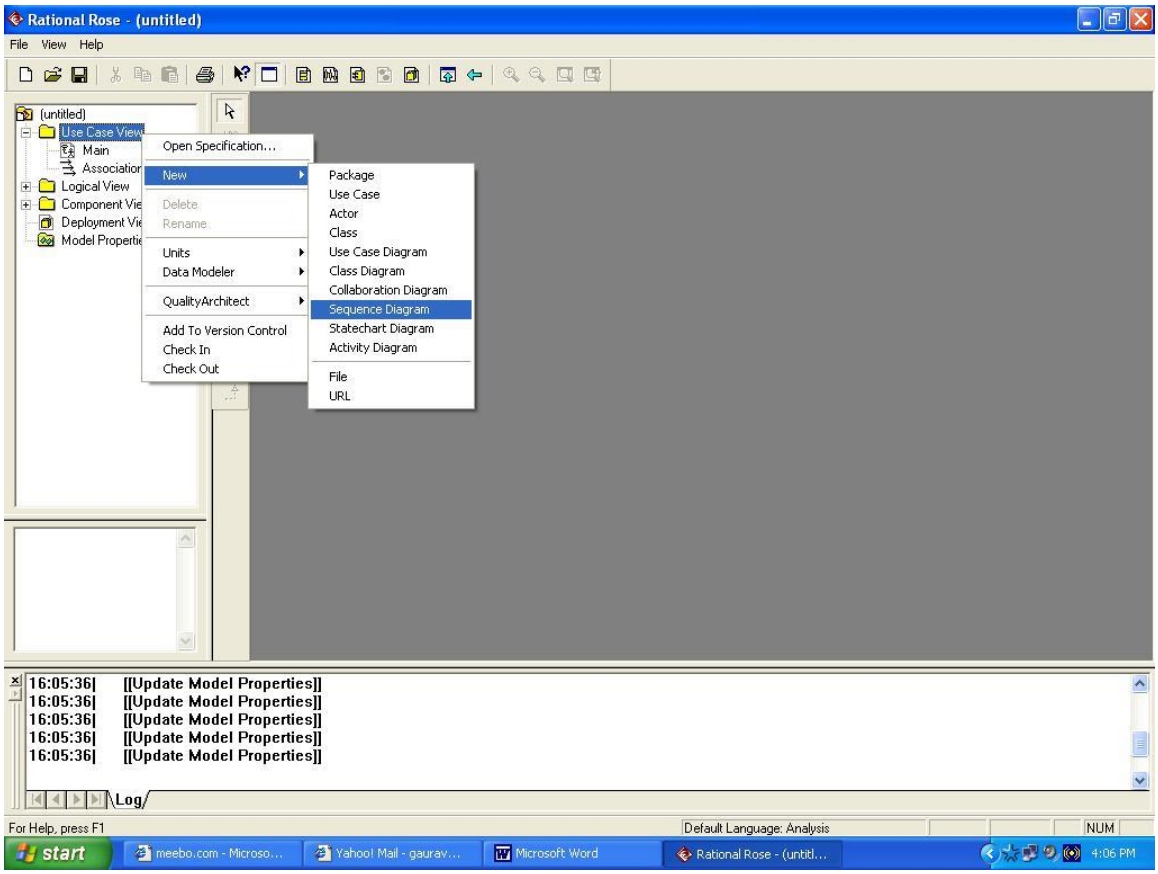
- Click on the File menu and select New.



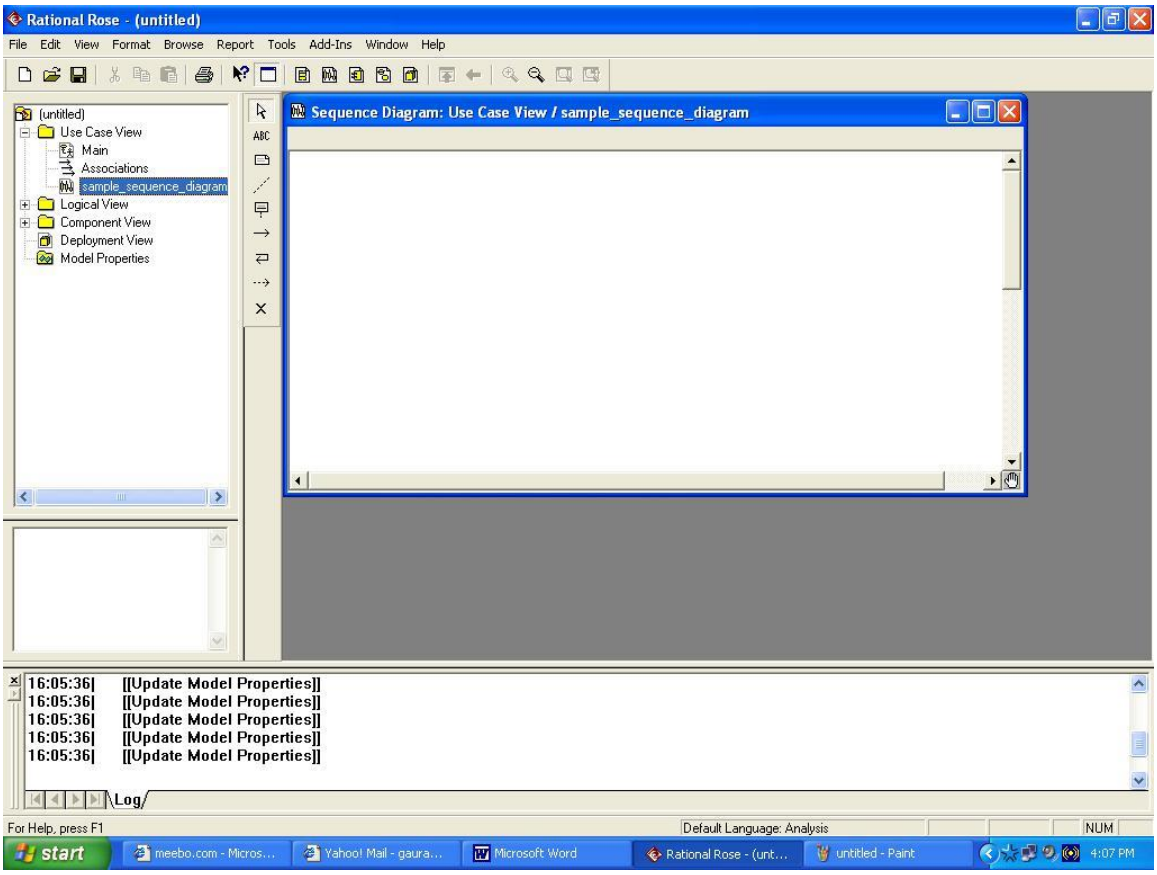
- Now from the Dialogue Box that appears ,select the language which you want to use for creating your model.



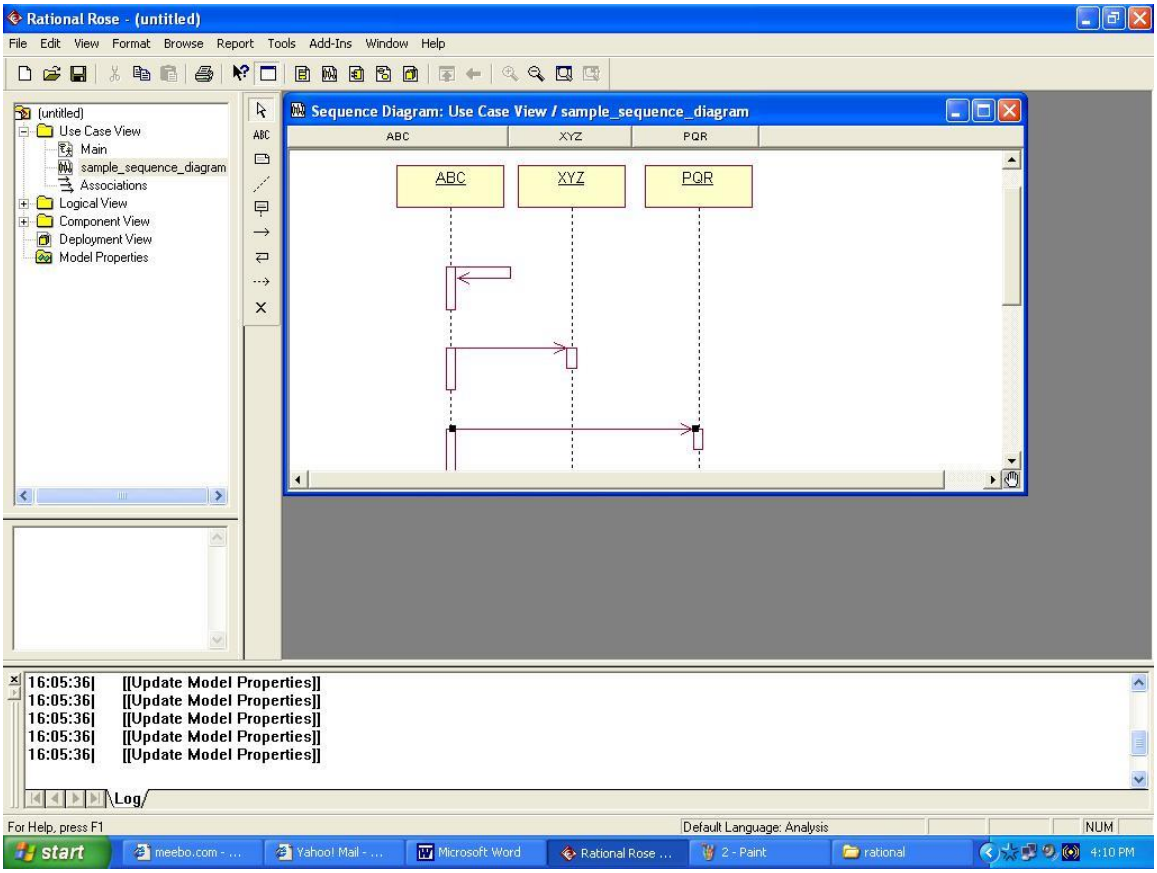
In the left hand side window of Rational Rose right click on ―Use Case view‖ and select New>Sequence Diagram



- Enter the name of new Sequence file in the space provided,and then click on that file name.

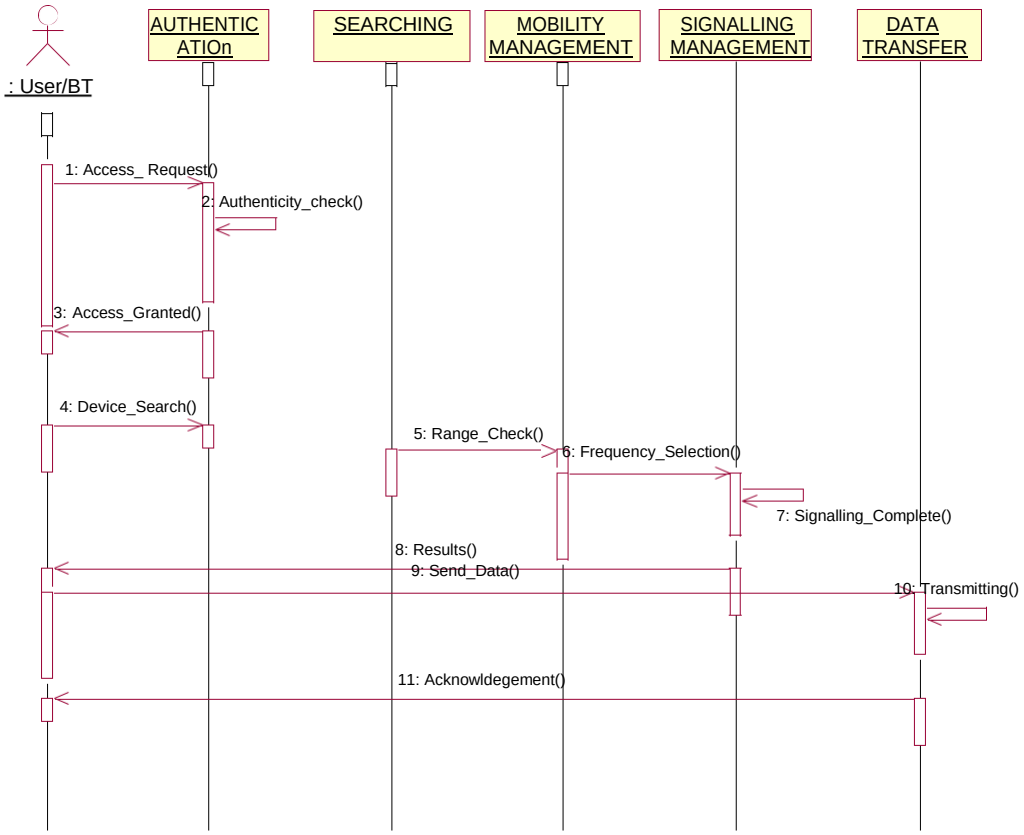


You can now use the window that appears on right hand side to draw your Sequence diagram using the buttons provided on the vertical toolbar.



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

Another example of a sequence diagram



Experiment No: 9

Aim:

Steps to draw the collaboration Diagram using Rational Rose.

Hardware Requirements:

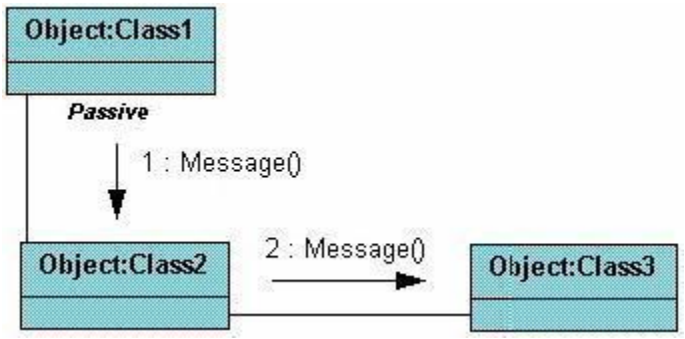
Pentium 4 processor (2.4 GHz), 128 Mb RAM, Standard keyboard and mouse, colored monitor.

Software Requirements:

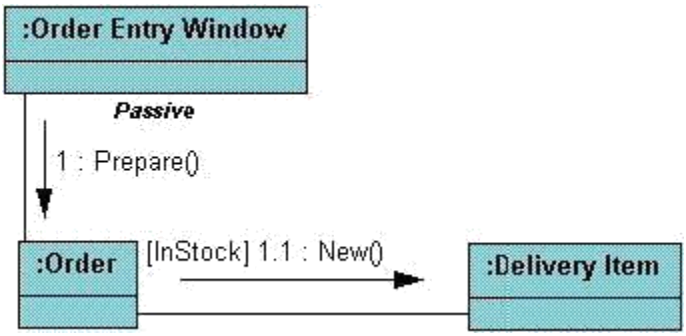
Rational Rose, Windows XP

THEORY

Collaboration diagrams are also 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.



The example below shows a simple collaboration diagram for the placing an order use case. This time the names of the objects appear after the colon, such as :Order Entry Window following the objectName:className naming convention. This time the class name is shown to demonstrate that all of objects of that class will behave the same way.


:Order Entry Window
Passive
1 : Prepare()
:Order
[InStock] 1.1 : New()
:Delivery Item

Experiment No: 10

AIM: To draw class diagram for any project.

REQUIREMENTS:

Hardware Interfaces

- Pentium(R) 4 CPU 2.26 GHz, 128 MB RAM
- Screen resolution of at least 800 x 600 required for proper and complete viewing of screens. Higher resolution would not be a problem.
- CD ROM Driver

Software Interfaces

- Any window-based operating system(Windows98/2000/XP/NT)
- IBM Rational Rose Software

THEORY:

A **class diagram** is a type of **static structure diagram** that describes the structure of a system by showing the system's classes, their attributes, and the relationships between the classes.

Class diagrams show the classes of the system, their inter-relationships, and the operations and attributes of the classes. Class diagrams are typically used, although not all at once, to:

- Explore domain concepts in the form of a domain model
- Analyze requirements in the form of a conceptual/analysis model
- Depict the detailed design of object-oriented or object-based software

A class model is comprised of one or more class diagrams and the supporting specifications that describe model elements including classes, relationships between classes, and interfaces. There are guidelines

1. **General issues**
2. Classes
3. **Interfaces**
4. Relationships

5. Inheritance

6. Aggregation and Composition

GENERAL GUIDELINES

Because class diagrams are used for a variety of purposes – from understanding requirements to describing your detailed design – it is needed to apply a different style in each circumstance. This section describes style guidelines pertaining to different types of class diagrams.

CLASSES

A class in the software system is represented by a box with the name of the class written inside it. A compartment below the class name can show the class's attributes (i.e. its properties). Each attribute is shown with at least its name, and optionally with its type, initial value, and other properties.

A class is effectively a template from which objects are created (instantiated). Classes define attributes, information that is pertinent to their instances, and operations, functionality that the objects support. Classes will also realize interfaces (more on this later).

Class diagrams are widely used to describe the types of objects in a system and their relationships. Class diagrams model class structure and contents using design elements such as classes, packages and objects. Class diagrams describe three different perspectives when designing a system, conceptual, specification, and implementation. These perspectives become evident as the diagram is created and help solidify the design.

INTERFACES

An interface is a collection of operation signature and/or attribute definitions that ideally defines a cohesive set of behaviors. **I**nterface a class or component must implement the operations and attributes defined by the interface. Any given class or component may implement zero or more interfaces and one or more classes or components can implement the same interface.

RELATIONSHIPS

A relationship is a general term covering the specific types of logical connections found on a class and object diagram.

Class diagrams also display relationships such as containment, inheritance, associations and others.

The association relationship is the most common relationship in a class diagram. The association shows the relationship between instances of classes.

Another common relationship in class diagrams is a generalization. A generalization is used when two classes are similar, but have some differences.

AGGREGATION

Aggregation is a variant of the "has a" or association relationship; composition is more specific than aggregation.

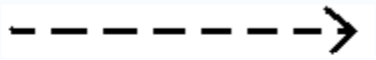
Aggregation occurs when a class is a collection or container of other classes, but where the contained classes do not have a strong **life cycle dependency** on the container--essentially, if the container is destroyed, its contents are not.

ASSOCIATION

Association are semantic connection between classes. When an association connects two classes , each class can send messages to other in a sequence or a collaboration diagram . Associations can be bidirectional or unidirectional.

DEPENDENCIES

Dependencies connect two clases . Dependencies are always unidirectional and show that one class, depends on the definitions in another class .



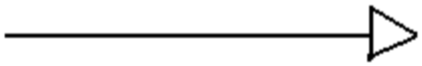
GENERALIZATION

The generalization relationship indicates that one of the two related classes (the supertype) is considered to be a more general form of the other (the subtype). In practice, this means that any instance of the subtype is also an instance of the supertype .

The generalization relationship is also known as the inheritance or "is a" relationship.

The supertype in the generalization relationship is also known as the "parent", superclass, base class, or base type.

The subtype in the generalization relationship is also known as the "child", subclass, derived class, derived type, inheriting class, or inheriting type.



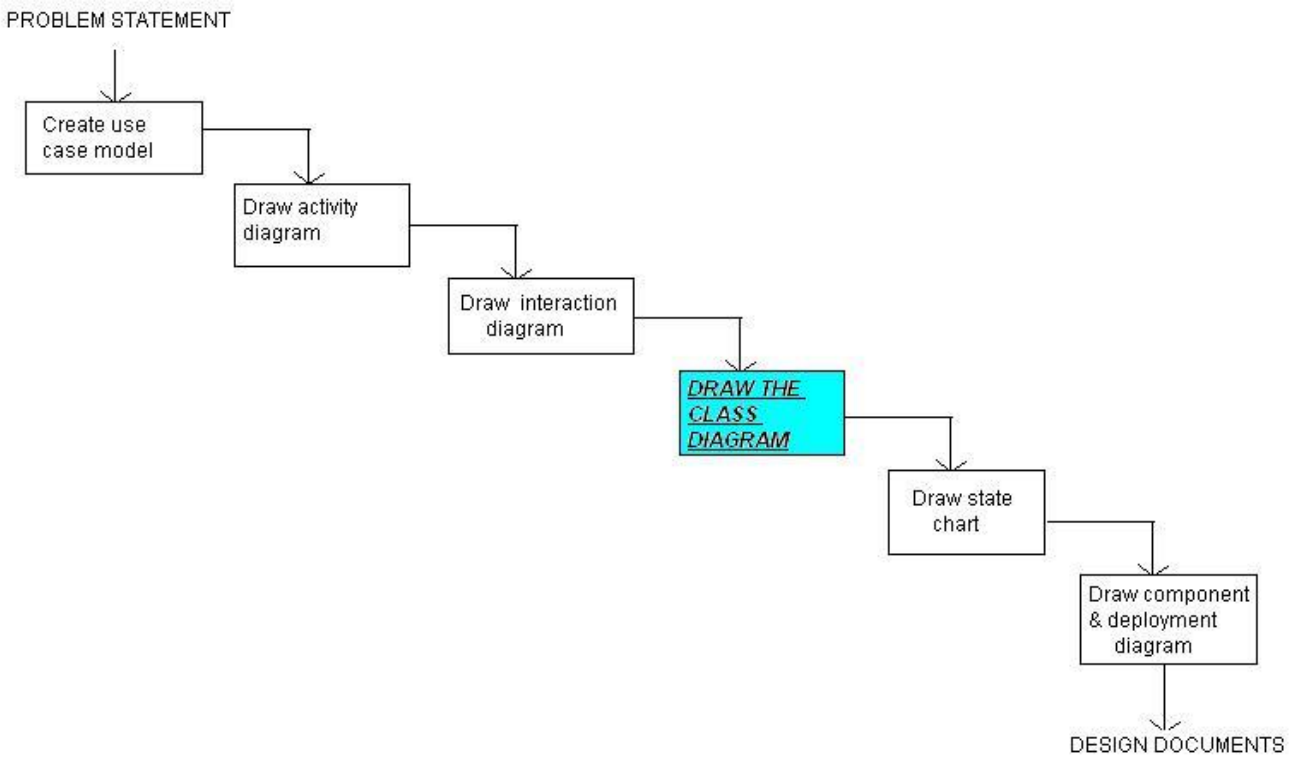
MULTIPLICITY

The association relationship indicates that (at least) one of the two related classes makes reference to the other.

HOW TO DRAW CLASS DIAGRAM

When designing classes the attributes and operations it will have are observed. Then determining how instances of the classes will interact with each other. These are the very first

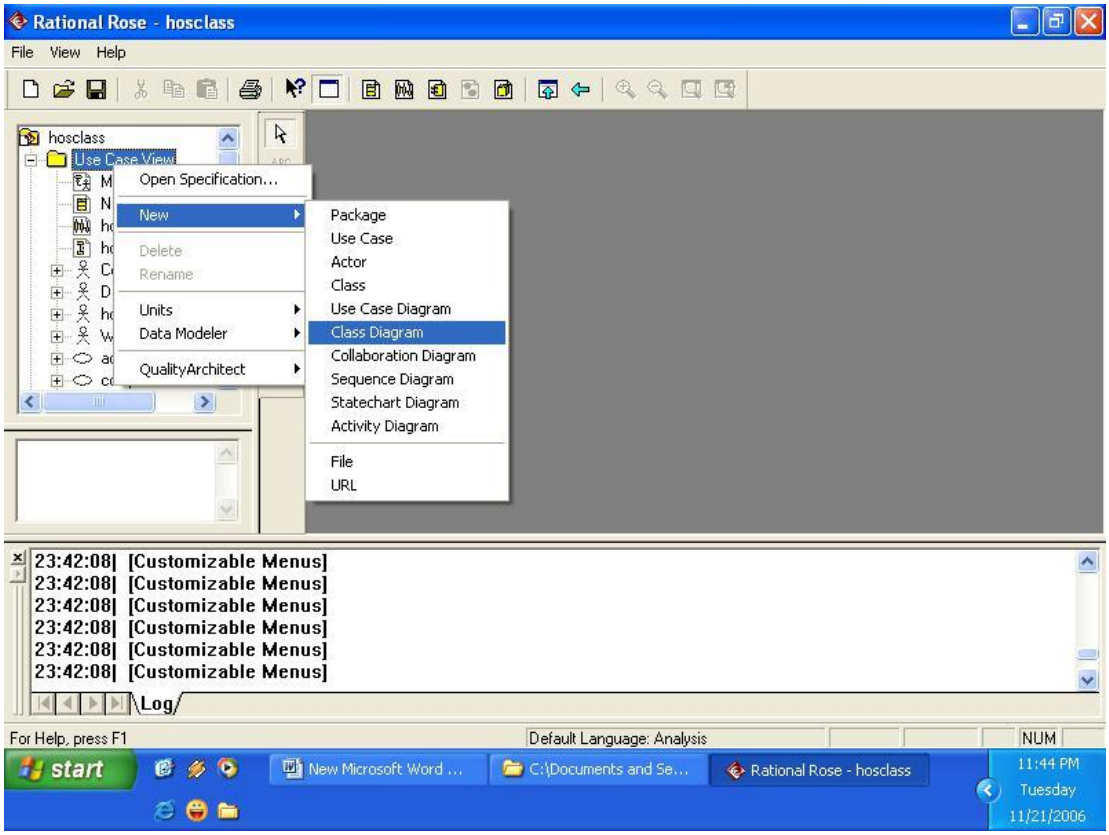
steps of many in developing a class diagram. However, using just these basic techniques one can develop a complete view of the software system. There are various steps in the analysis and design of an object oriented system.



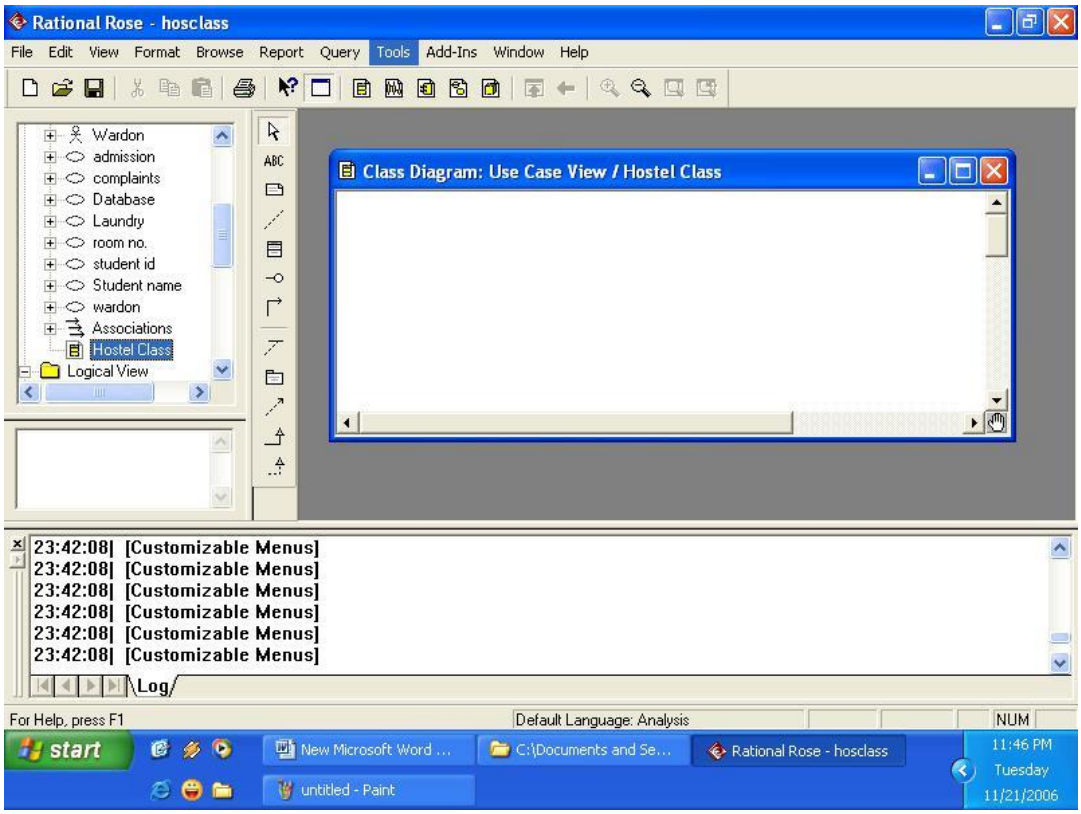
STEPS FOR ANALYSIS AND DESIGN

STEPS FOR DRAWING CLASS DIAGRAM

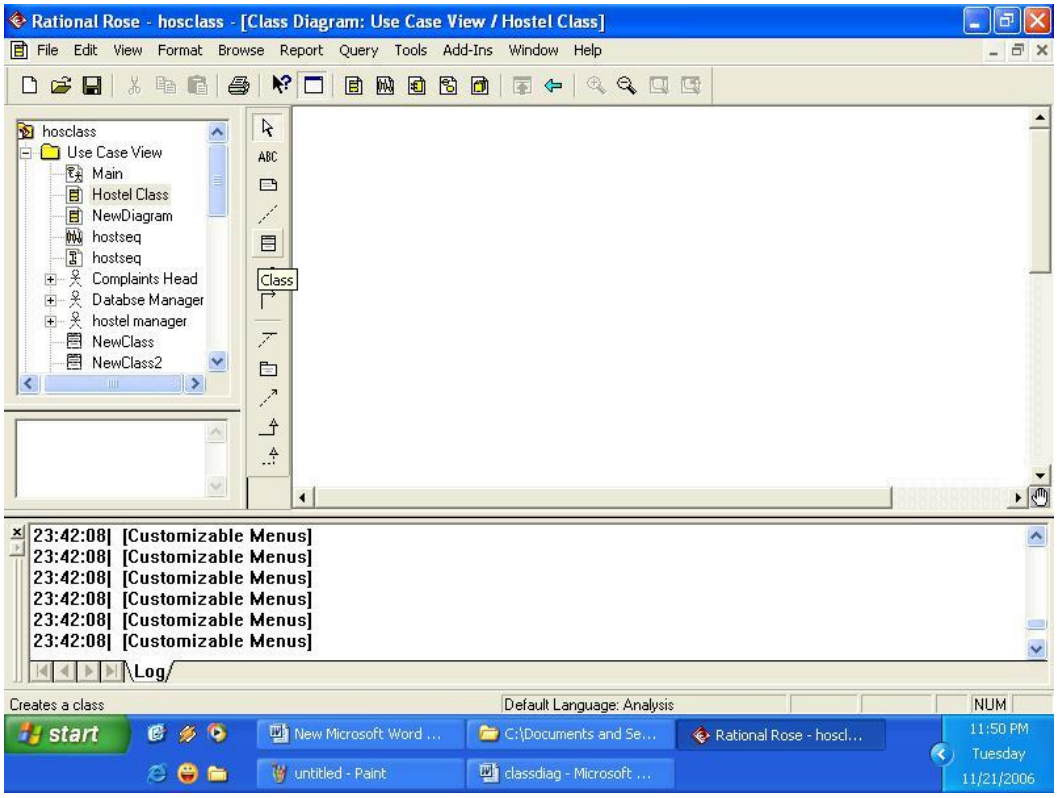
1. After completing the sequence diagrams and collaboration diagram which are a part of the interaction diagrams. In Rational Rose, right click on the ―Use Case View‖ and select new class diagram.



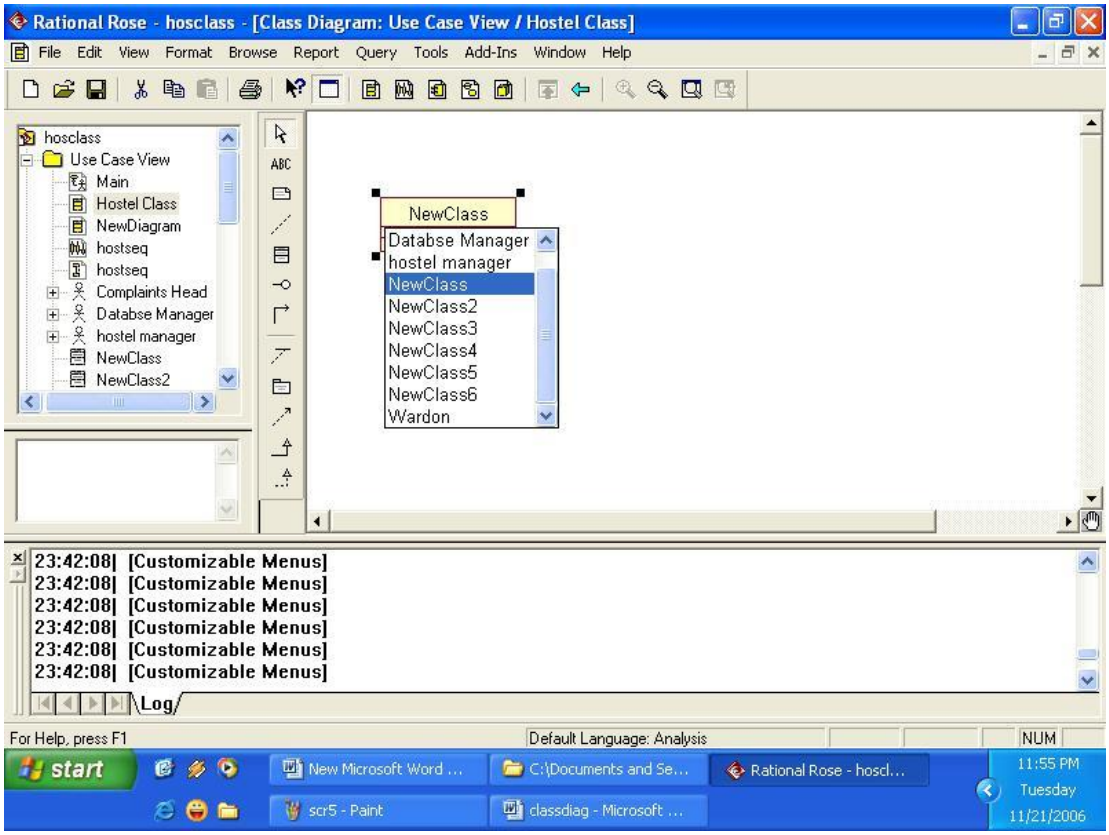
2. Enter the class name (here ―Hostel Class‖).



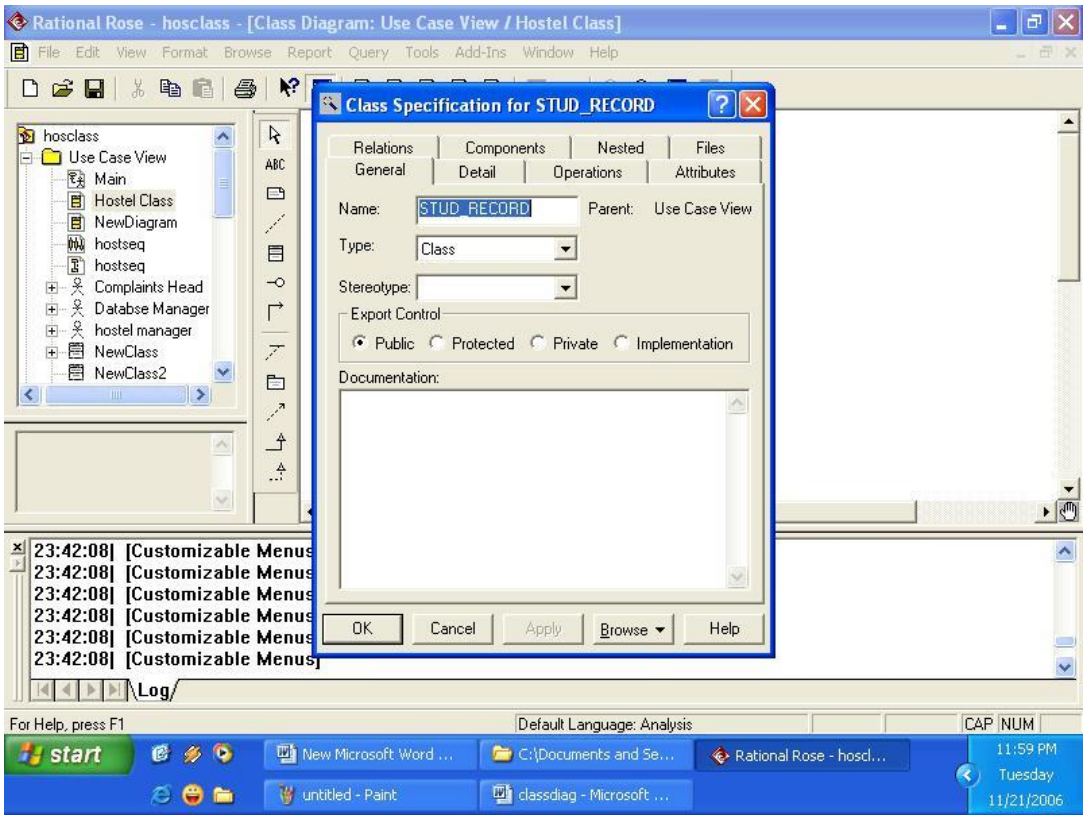
3. Click the cursor on the block representing class from the table of predefined symbols into the screen



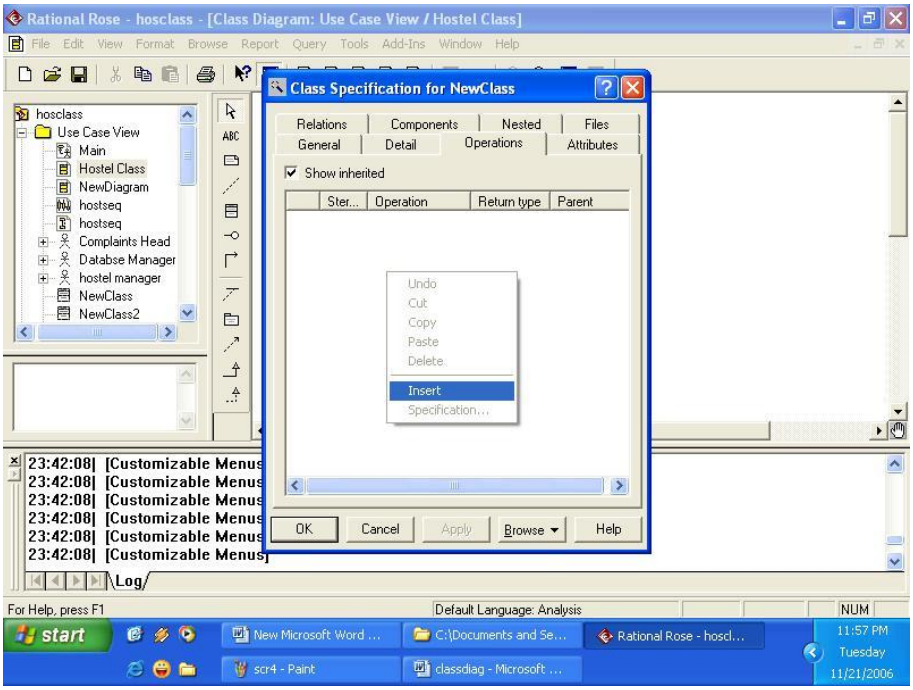
4. Select a new Class



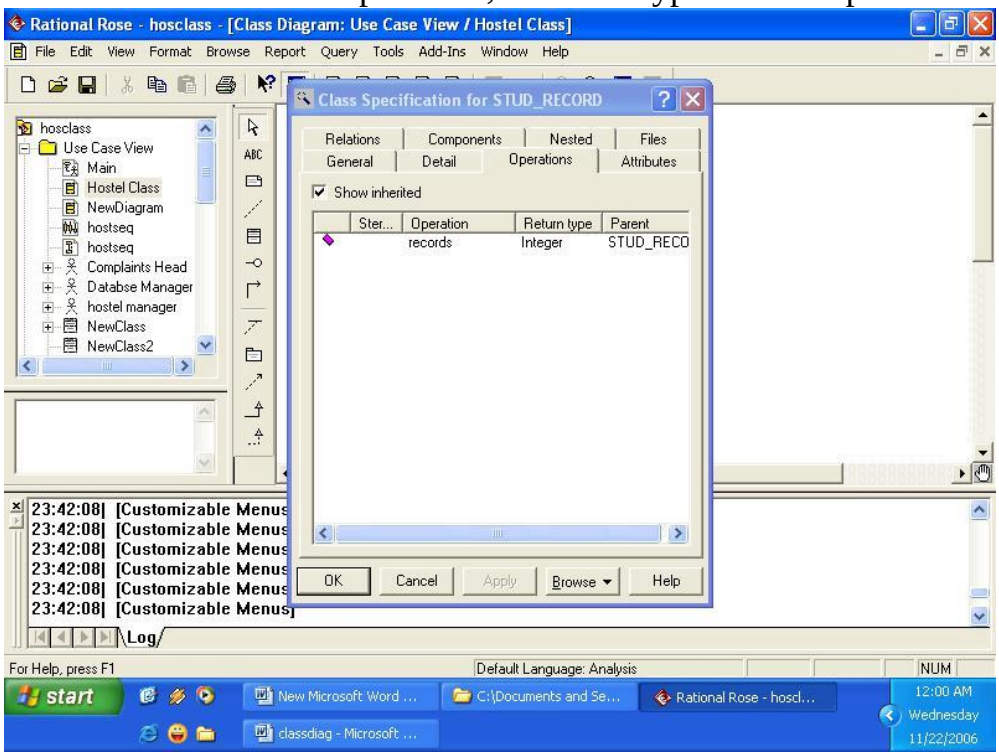
1. Double click on the class formed to enter the class name in the general field .



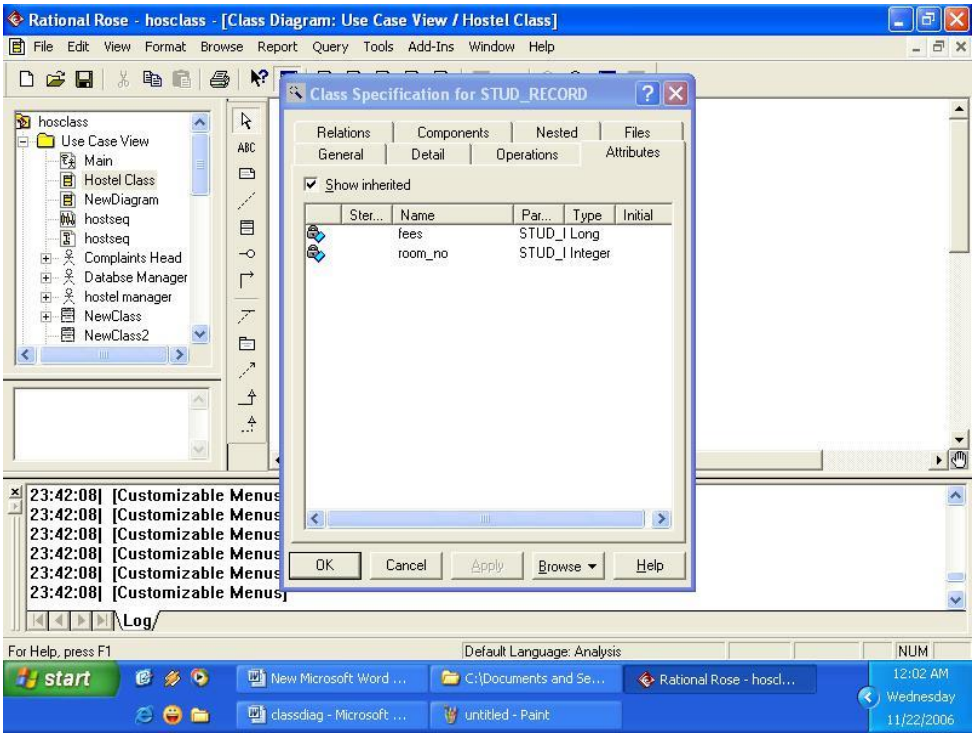
2. In the operation field right click and select the inset option to add class operations or functions.



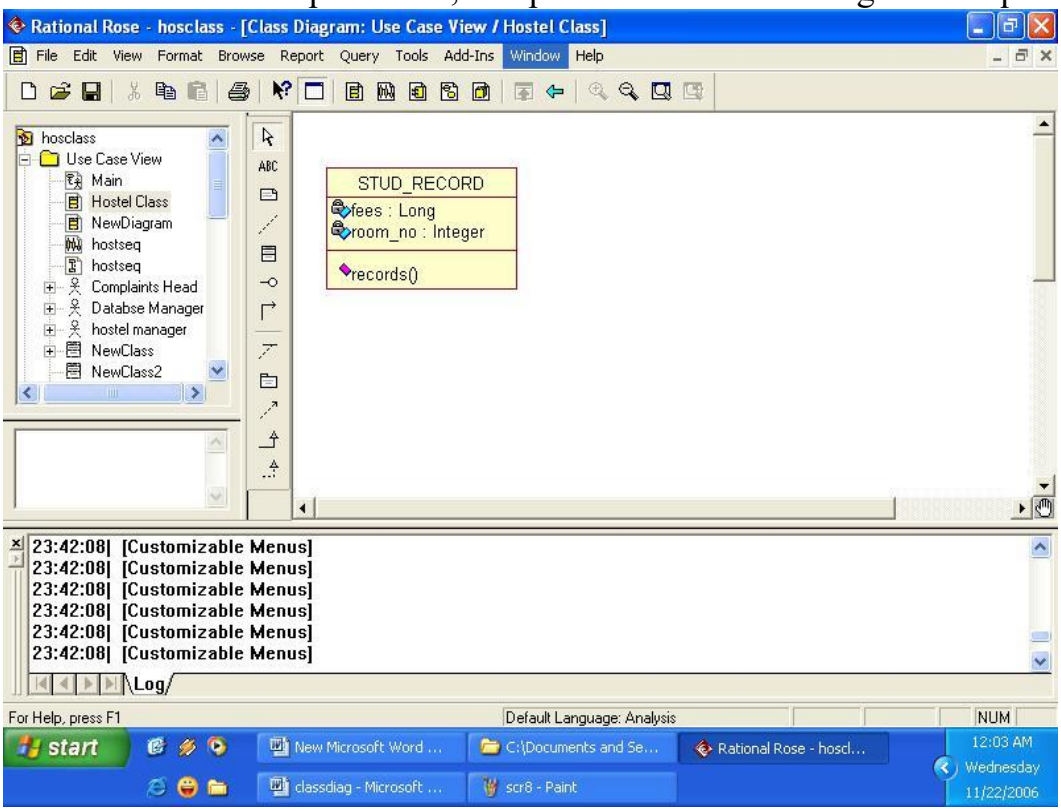
3. Input the name of the new operation , its return type in the respective columns.



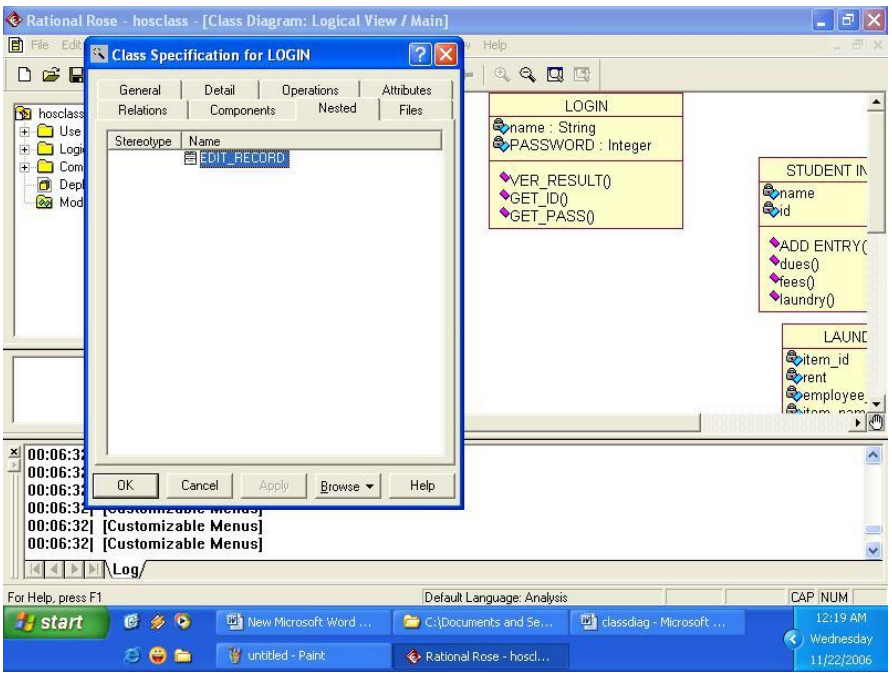
7. In the attribute field, enter the attribute names , their type and the parent class in the respective columns.



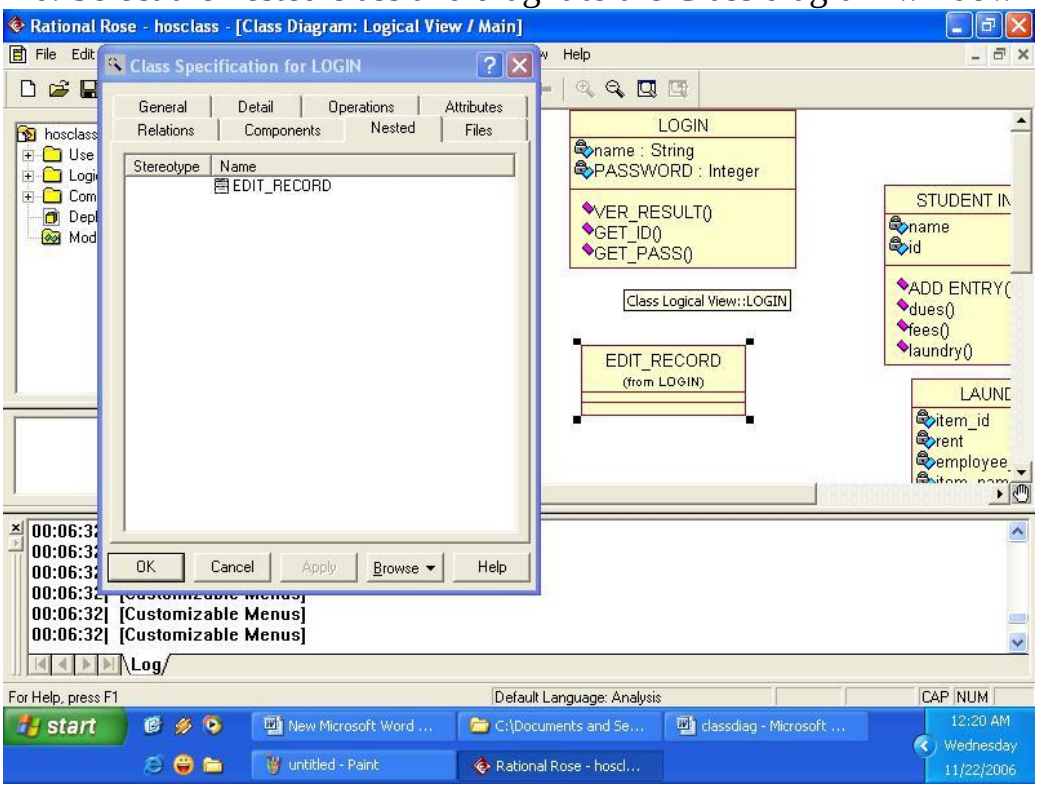
8. Enter all the attributes and operations , and press —OK‖ button to get the required class.



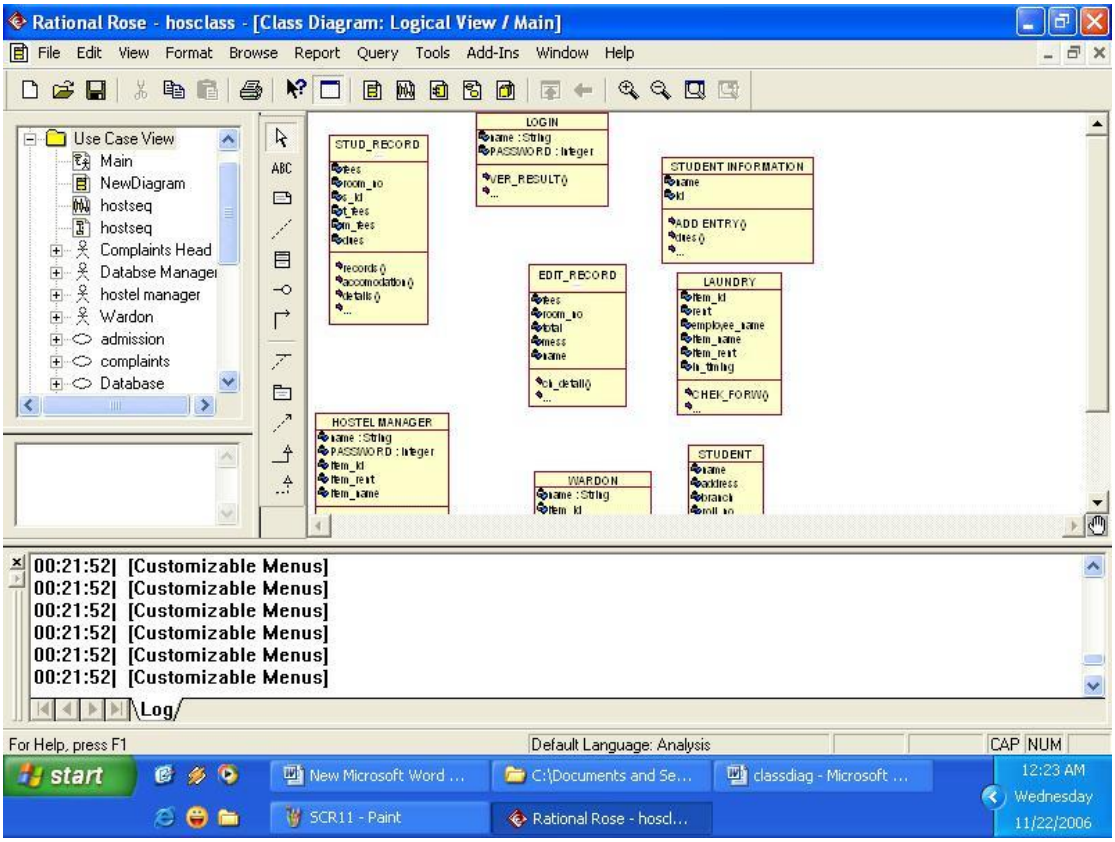
9. While building the classes of the system if you want to make nested class in some main class(here ―LOGIN‖ class), then insert classes in the _Nested‘ field of class specification(like the ―EDIT_RECORD‖ class)



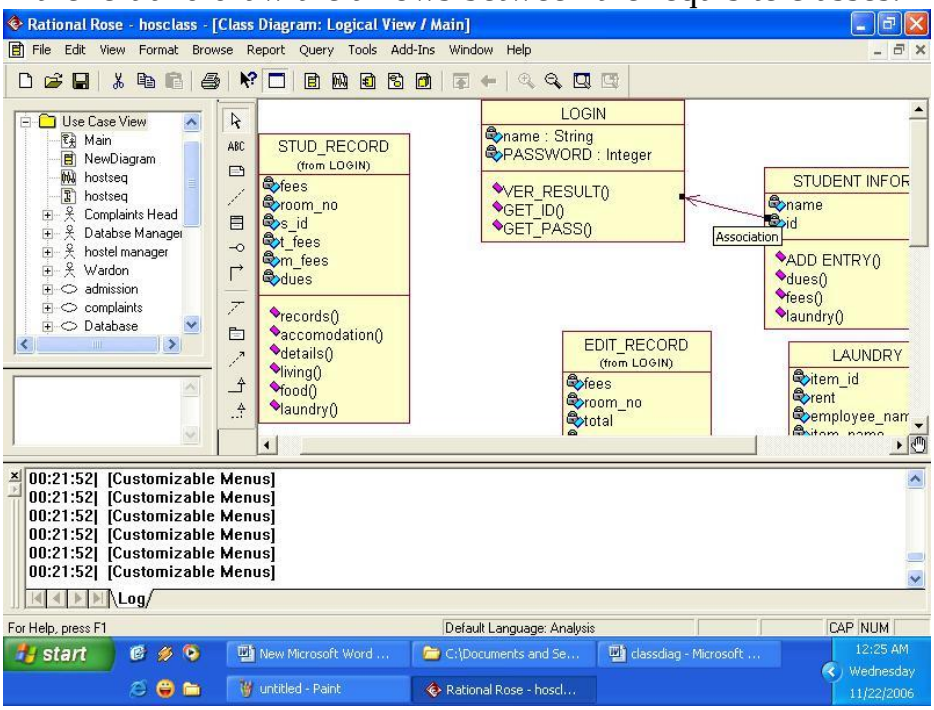
10. Select the nested class and drag it to the Class diagram window



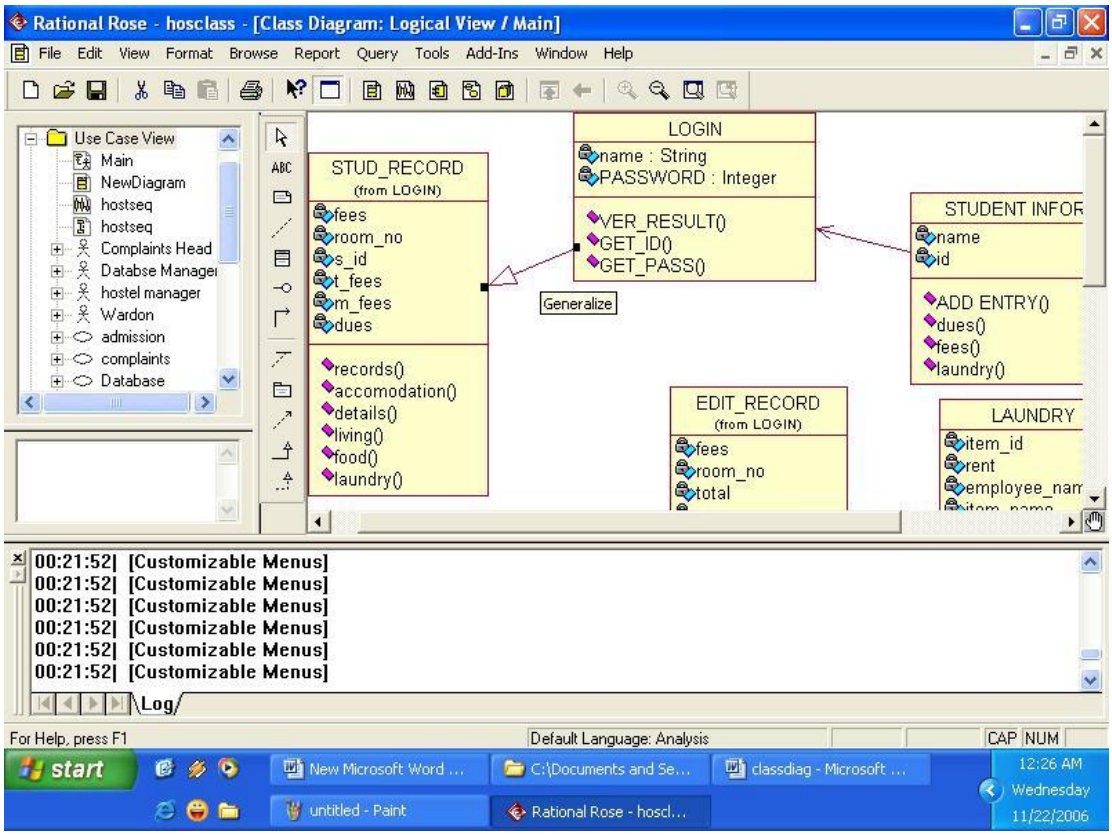
11. All the required classes were built completely with there operations, attributes and nested classes , into the class diagram



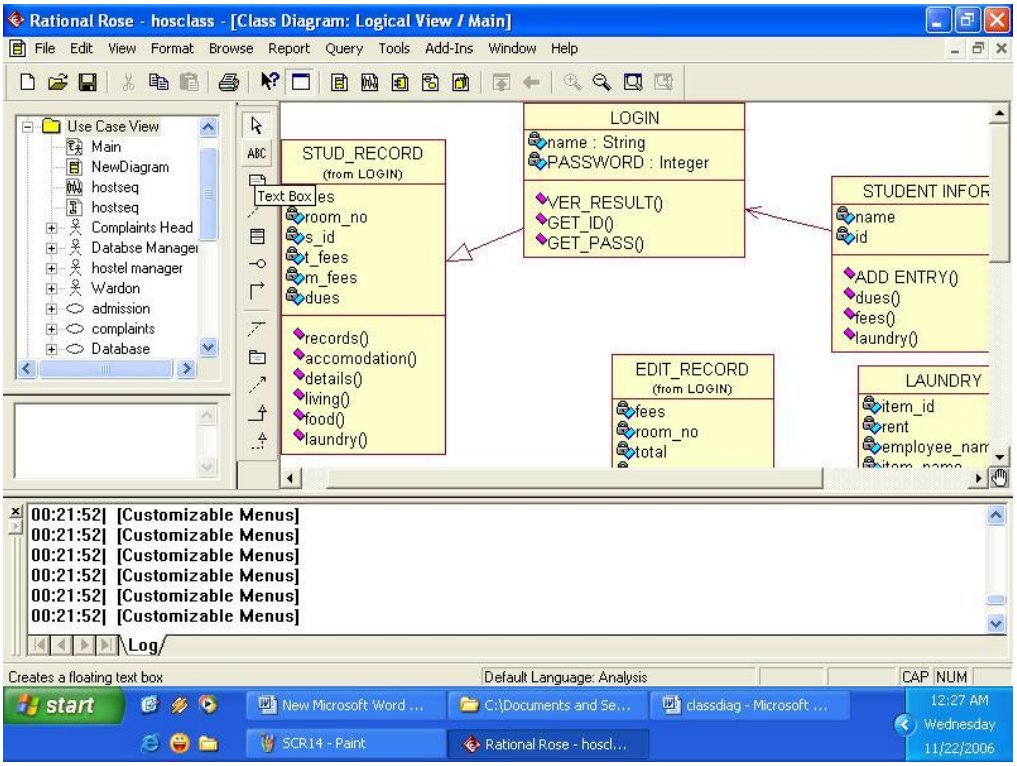
12. Now we want to show the relationships between the various classes. To show an ASSOCIATION relation select a block named _association‘ from the blocks to the left and draw the arrows between the requisite classes.



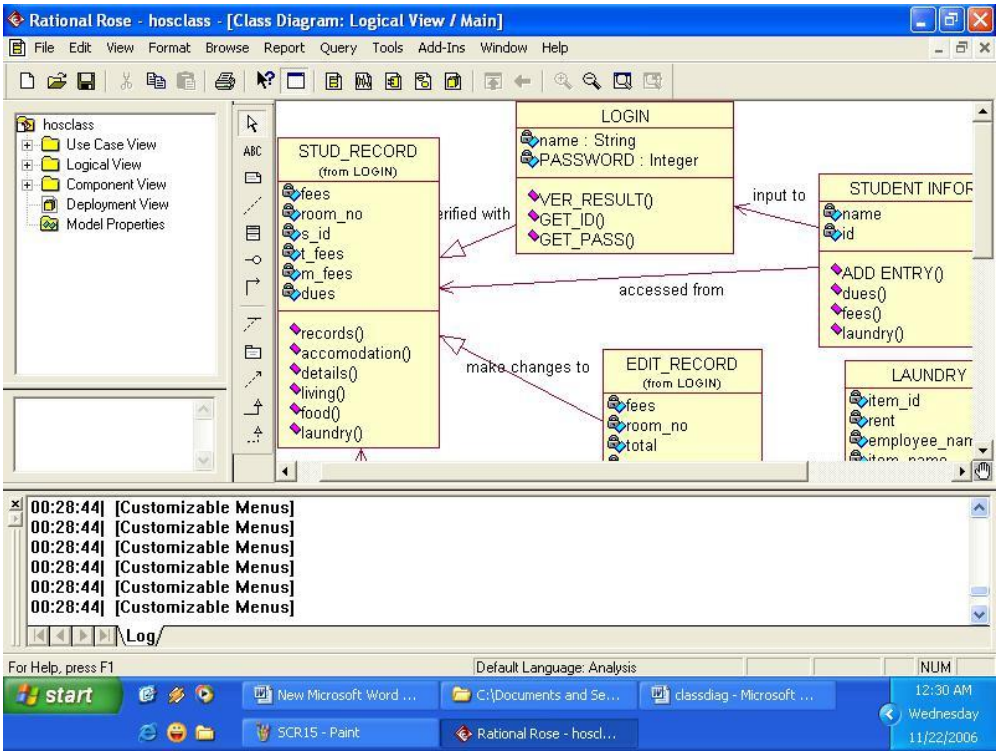
13. To show a GENERALIZATION or inheritance relation select a block named _generalization‘ from the blocks to the left and draw the arrows between the requisite classes.



14. Select the _text box‘ block from the blocks field to describe any relation with the help of text.



15. Enter text by placing text boxes over the various relationship arrows



Conclusion : The class diagram was made successfully by following the above steps .