

**Department of Computer Science
&
Engineering**

LAB MANUAL

Session 2016-2017

Subject Name :- Software Engineering Lab
Subject Code :- CS-217
Branch :- CSE
Year :- 2nd
Semester :- 4th

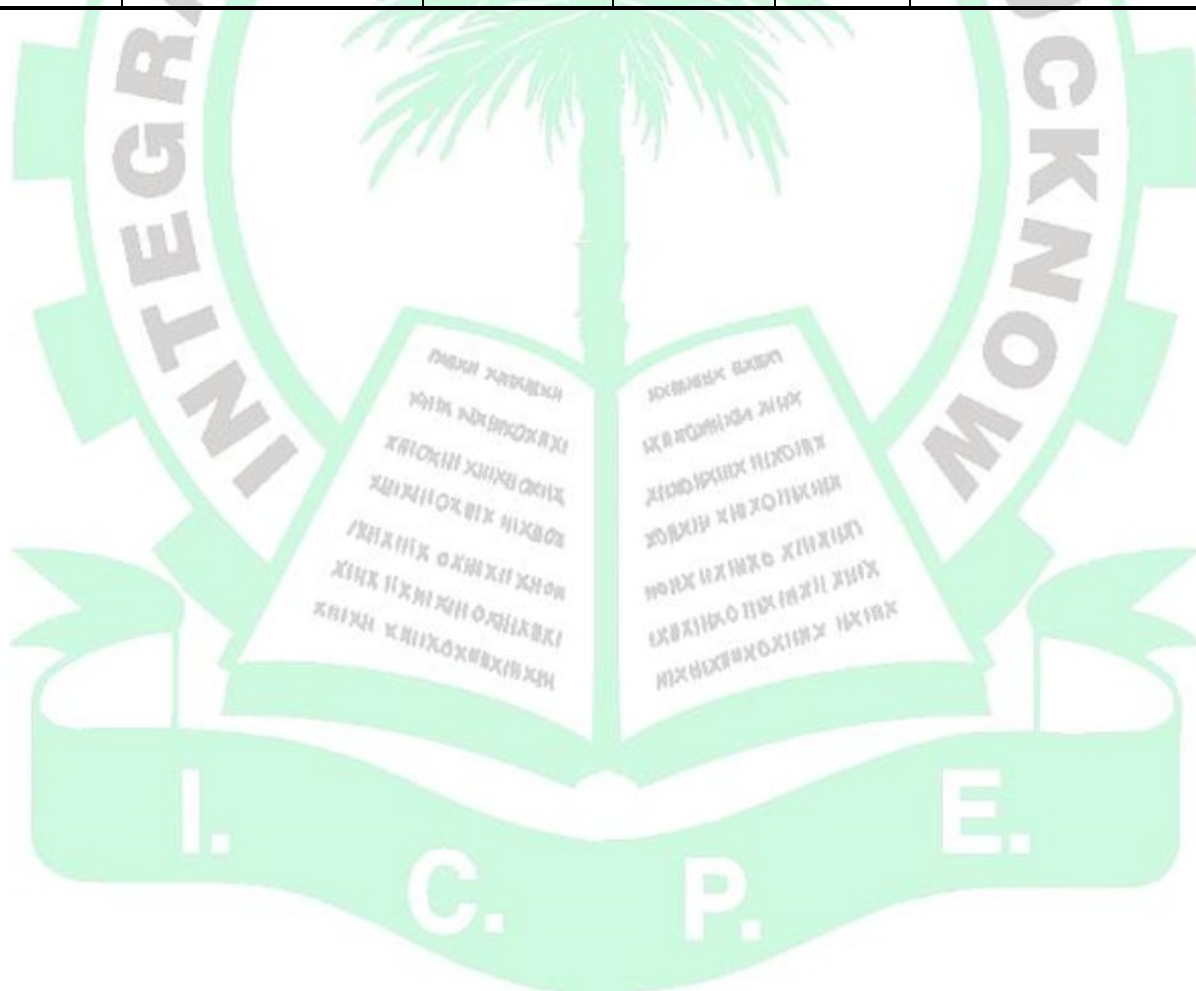


INTEGRAL UNIVERSITY LUCKNOW

Dsauli, Kursi Road, PO Basha-226026

Revision History

REVISION HISTORY					
REVISION #	AUTHOR(S)	REVIEWER(S)	DATE OF RELEASE	OWNER	SUMMARY OF CHANGES
Revision - 1	Mr. Balmukund Maurya Mr. Tabrez Khan	Internal Reviewers	01/01/2014	HOD - CSE	Initial Release
Revision -2		Departmental QAC	31/01/2014	HOD - CSE	Major revisions made. Incorporated review comments from departmental QAC into Revision-2
Revision -3		External Reviewers	28/02/2013	HOD - CSE	
			02/01/2017	HOD - CSE	

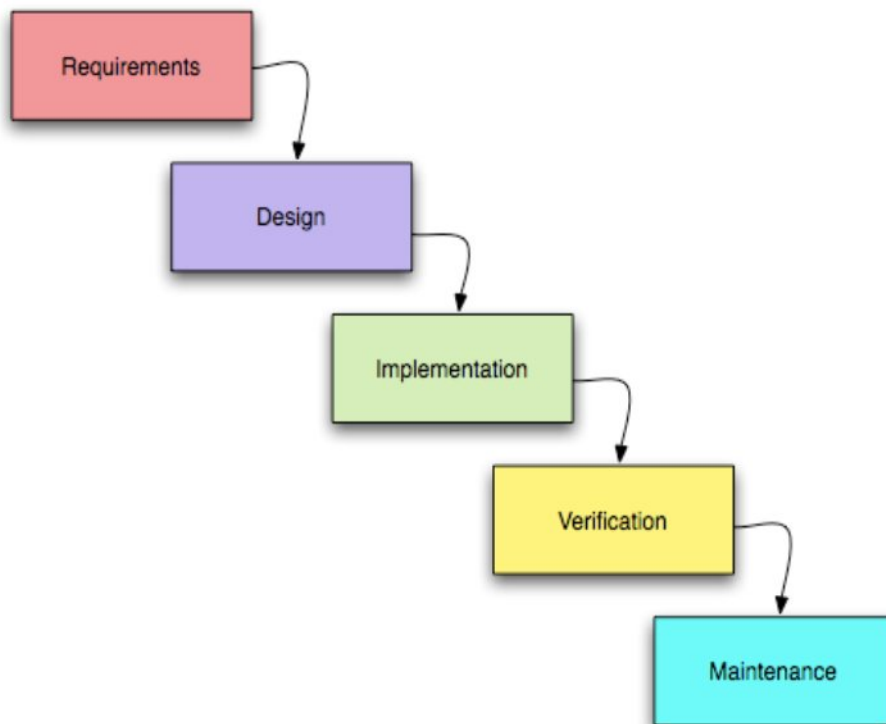


List of Lab Exercises

Week	Experiment No.	Name of the Experiments	Page No.
Week 1		Introduction	4
	1	Identifying the Requirements from Problem Statements	5
Week-2			
	2	Introduction (MS Project)	10
		Task Scheduling (MS Project)	
Week-3			
	3	Modeling Data Flow Diagrams	14
Week-4			
	4	E-R Modeling from the Problem Statements.	19
Week-5			
	5	General study of UML	28
Week-6			
	6	UML Use Case Diagram	38
Week-7			
	7	UML Class Diagram	50
	8	Study of Object Diagram	52
Week-8			
	9	UML Deployment diagram	54
	10	UML Activity Diagram	57
Week-9			
	11	UML State chart Diagram	59
Week-10			
	12	Automated Functional Testing Using IBM Rational Robot	

Introduction

- Software engineering is the application of a systematic, disciplined, quantifiable approach to the development, operation, and maintenance of software, and the study of these approaches; that is, the application of engineering to software. The term software engineering first appeared in the 1968 NATO Software Engineering Conference and was meant to provoke thought regarding the current "software crisis" at the time. Since then, it has continued as a profession and field of study dedicated to creating software that is of higher quality, more affordable, maintainable, and quicker to build. Since the field is still relatively young compared to its sister fields of engineering, there is still much debate around what software engineering actually is, and if it conforms to the classical definition of engineering. It has grown organically out of the limitations of viewing software as just computer programming.
- What is a software life cycle model?
A software life cycle model is either a descriptive or prescriptive characterization of how software is or should be developed. A descriptive model describes the history of how a particular software system was developed. Descriptive models may be used as the basis for understanding and improving software development processes, or for building empirically grounded prescriptive models



Experiment-1

OBJECTIVE

- Identifying the Requirements from Problem Statements
- Learn how to prepare project plans

Requirements identification is the first step of any software development project. Until the requirements of a client have been clearly identified, and verified, no other task (design, coding, testing) could begin. Usually business analysts having domain knowledge on the subject matter discuss with clients and decide what features are to be implemented.

In this experiment we will learn how to identify functional and non-functional requirements from a given problem statement. Functional and non-functional requirements are the primary components of a Software Requirements Specification.

After completing this experiment you will be able to:

- Identify ambiguities, inconsistencies and incompleteness from a requirements specification
- Identify and state functional requirements
- Identify and state non-functional requirements

Requirements

Sommerville defines "requirement" [1] as a specification of what should be implemented. Requirements specify how the target system should behave. It specifies what to do, but not how to do. Requirements engineering refers to the process of understanding what a customer expects from the system to be developed, and to document them in a standard and easily readable and understandable format. This documentation will serve as reference for the subsequent design, implementation and verification of the system.

It is necessary and important that before we start planning, design and implementation of the software system for our client, we are clear about its requirements. If we don't have a clear vision of what is to be developed and what all features are expected, there would be serious problems, and customer dissatisfaction as well.

Characteristics of Requirements

Requirements gathered for any new system to be developed should exhibit the following three properties:

- **Unambiguity:** There should not be any ambiguity what a system to be developed should do. For example, consider you are developing a web application for your client. The client requires that enough number of people should be able to access the application simultaneously. What's the "enough number of people"? That could mean 10 to you, but, perhaps, 100 to the client. There's an ambiguity.
- **Consistency:** To illustrate this, consider the automation of a nuclear plant. Suppose one of the clients say that if the radiation level inside the plant exceeds R1, all reactors should be shut down. However, another person from the client side suggests that the threshold radiation level should be R2. Thus, there is an inconsistency between the two end users regarding what they consider as threshold level of radiation.
- **Completeness:** A particular requirement for a system should specify what the system should do and also what it should not. For example, consider a software to be developed for ATM. If a customer enters an amount greater than the maximum

permissible withdrawal amount, the ATM should display an error message, and it should not dispense any cash.

Categorization of Requirements

Based on the target audience or subject matter, requirements can be classified into different types, as stated below:

- **User requirements:** They are written in natural language so that both customers can verify their requirements have been correctly identified
- **System requirements:** They are written involving technical terms and/or specifications, and are meant for the development or testing teams

Requirements can be classified into two groups based on what they describe:

- **Functional requirements (FRs):** These describe the functionality of a system -- how a system should react to a particular set of inputs and what should be the corresponding output.
- **Non-functional requirements (NFRs):** They are not directly related what functionalities are expected from the system. However, NFRs could typically define how the system should behave under certain situations. For example, a NFR could say that the system should work with 128MB RAM. Under such condition, a NFR could be more critical than a FR.

Non-functional requirements could be further classified into different types like:

- **Product requirements:** For example, a specification that the web application should use only plain HTML, and no frames
- **Performance requirements:** For example, the system should remain available 24x7
- **Organizational requirements:** The development process should comply to SEI CMM level 4

Functional Requirements

Identifying Functional Requirements

Given a problem statement, the functional requirements could be identified by focusing on the following points:

- Identify the high level functional requirements simply from the conceptual understanding of the problem. For example, a Library Management System, apart from anything else, should be able to issue and return books.
- Identify the cases where an end user gets some meaningful work done by using the system. For example, in a digital library a user might use the "Search Book" functionality to obtain information about the books of his interest.
- If we consider the system as a black box, there would be some inputs to it, and some output in return. This black box defines the functionalities of the system. For example, to search for a book, user gives title of the book as input and get the book details and location as the output.
- Any high level requirement identified could have different sub-requirements. For example, "Issue Book" module could behave differently for different class of users, or for a particular user who has issued the book thrice consecutively.

Preparing Software Requirements Specifications

Once all possible FRs and non-FRs have been identified, which are complete, consistent, and non-ambiguous, the Software Requirements Specification (SRS) is to be prepared. IEEE provides a template [\[iv\]](#), also available [here](#), which could be used for this purpose. The SRS is prepared by the service provider, and verified by its client. This document serves as a legal agreement between the client and the service provider. Once the concerned system has been developed and deployed, and a proposed feature was not found to be present in the system, the client can point this out from the SRS. Also, if after delivery, the client says a new feature

is required, which was not mentioned in the SRS, the service provider can again point to the SRS. The scope of the current experiment, however, doesn't cover writing a SRS.

Case Study

1 : A Library Information System for SE VLabs Institute

The SE VLabs Institute has been recently setup to provide state-of-the-art research facilities in the field of Software Engineering. Apart from research scholars (students) and professors, it also includes quite a large number of employees who work on different projects undertaken by the institution.

As the size and capacity of the institute is increasing with the time, it has been proposed to develop a Library Information System (LIS) for the benefit of students and employees of the institute. LIS will enable the members to borrow a book (or return it) with ease while sitting at his desk/chamber. The system also enables a member to extend the date of his borrowing if no other booking for that particular book has been made. For the library staff, this system aids them to easily handle day-to-day book transactions. The librarian, who has administrative privileges and complete control over the system, can enter a new record into the system when a new book has been purchased, or remove a record in case any book is taken off the shelf. Any non-member is free to use this system to browse/search books online. However, issuing or returning books is restricted to valid users (members) of LIS only.

The final deliverable would be a web application (using the recent HTML 5), which should run only within the institute LAN. Although this reduces security risk of the software to a large extent, care should be taken no confidential information (eg., passwords) is stored in plain text.

Identification of functional requirements

The above problem statement gives a brief description of the proposed system. From the above, even without doing any deep analysis, we might easily identify some of the basic functionality of the system:

- **New user registration:** Any member of the institute who wishes to avail the facilities of the library has to register himself with the Library Information System. On successful registration, a user ID and password would be provided to the member. He has to use this credentials for any future transaction in LIS.
- **Search book:** Any member of LIS can avail this facility to check whether any particular book is present in the institute's library. A book could be searched by its:
 - Title
 - Authors name
 - Publisher's name
- **User login:** A registered user of LIS can login to the system by providing his employee ID and password as set by him while registering. After successful login, "Home" page for the user is shown from where he can access the different functionalities of LIS: search book, issue book, return book, reissue book. Any employee ID not registered with LIS cannot access the "Home" page -- a login failure message would be shown to him, and the login dialog would appear again. This same thing happens when any registered user types in his password wrong. However, if incorrect password has been provided for three time consecutively, the security question for the user (specified while registering) with an input box to answer it are also shown. If the user can answer the security question correctly, a new password would be sent to his email address. In case the user fails to answer the security question correctly, his LIS account would be blocked. He needs to contact with the administrator to make it active again.

- **Issue book:** Any member of LIS can issue a book against his account provided that:
 - The book is available in the library i.e. could be found by searching for it in LIS
 - No other member has currently issued the book
 - Current user has not issued the maximum number of books that can

If the above conditions are met, the book is issued to the member.

Note that this FR would remain **incomplete** if the "maximum number of books that can be issued to a member" is not defined. We assume that this number has been set to four for students and research scholars, and to ten for professors.

Once a book has been successfully issued, the user account is updated to reflect the same.

- **Return book:** A book is issued for a finite time, which we assume to be a period of 20 days. That is, a book once issued should be returned within the next 20 days by the corresponding member of LIS. After successful return of a book, the user account is updated to reflect the same.
- **Reissue book:** Any member who has issued a book might find that his requirement is not over by 20 days. In that case, he might choose to reissue the book, and get the permission to keep it for another 20 days. However, a member can reissue any book at most twice, after which he has to return it. Once a book has been successfully reissued, the user account is updated to reflect the information.

In a similar way we can list other functionality offered by the system as well. However, certain features might not be evident directly from the problem system, but which, nevertheless, are required. One such functionality is "User Verification". The LIS should be able to judge between a registered and non-registered member. Most of the functionality would be available to a registered member. The "New User Registration" would, however, be available to non-members. Moreover, an already registered user shouldn't be allowed to register himself once again.

Having identified the (major) functional requirements, we assign an identifier to each of them [v] for future reference and verification. Following table shows the list:

#	Requirement	Priority
R1	New user registration	High
R2	User Login	High
R3	Search book	High
R4	Issue book	High
R5	Return book	High
R6	Reissue book	Low

Identification of non-functional requirements

Having talked about functional requirements, let's try to identify a few non-functional requirements.

- **Performance Requirements:**
 - This system should remain accessible 24x7
 - At least 50 users should be able to access the system altogether at any given time
- **Security Requirements:**
 - This system should be accessible only within the institute LAN

- The database of LIS should not store any password in plain text -- a hashed value has to be stored
- **Software Quality Attributes**
- **Database Requirements**
- **Design Constraints:**
 - The LIS has to be developed as a web application, which should work with Firefox 5, Internet Explorer 8, Google Chrome 12, Opera 10
 - The system should be developed using HTML 5

Once all the functional and non-functional requirements have been identified, they are documented formally in SRS, which then serves as a legal agreement.

Exercise:

1. Consider the problem statement for an "Online Auction System" to be developed:

New users can register to the system through an online process. By registering a user agrees to abide by different pre-defined terms and conditions as specified by the system. Any registered user can access the different features of the system authorized to him / her, after he authenticates himself through the login screen. An authenticated user can put items in the system for auction. Authenticated users can place bid for an item. Once the auction is over, the item will be sold to the user placing the maximum bid. Payments are to be made by third party payment services, which, of course, is guaranteed to be secure. The user selling the item will be responsible for it's shipping. If the seller thinks he's getting a good price, he can, however, sell the item at any point of time to the maximum bidder available.

Bibliography

Lecture on "System Analysis and Design", NPTEL

When Telepathy Won't Do: Requirements Engineering Key Practices

Requirements Analysis: Process of requirements gathering and requirement definition

IEEE Recommended Practice for Software Requirements Specifications

Requirements Trace-ability and Use Cases

Experiment - 2

OBJECTIVE

- Task Scheduling (MS PROJECT)
- Gain understanding of project management and planning.
- Learn how to prepare project plans.
- Getting familiar with MS Project.

INTRODUCTION

Project management is the process of planning and controlling the development of a system within a specified timeframe at a minimum cost with the right functionality.

The four major component of project planning is :

Tasks: They are a division of all the work that needs to be completed in order to accomplish the project goals.

Scope: of any project is a combination of all individual tasks and their goals.

Resources: can be people, equipment, materials or services that are needed to complete various tasks.

Tools Used in the Lab-Microsoft Project

Microsoft Project is the clear market leader among desktop project management applications.

Project Work Plan

Prepare a list of all tasks in the work breakdown structure, plus:

- Duration of task.
- Current task status.
- Task dependencies.
- Key milestone dates.

Tracking Progress

- Gantt Chart:
- Bar chart format.
- Useful to monitor project status at any point in time.
- PERT Chart:
- Flowchart format.
- Illustrate task dependencies and critical path.

STARTING A NEW PROJECT

- You can create a Project from a Template File by choosing File > New from the menu. In the New File dialog box that opens, select the Project Templates tab and select the template that suits your project best and click OK. (You may choose a Blank Project Template and customize it)
- Once a new Project page is opened, the Project Information dialog box opens.

Project Information for 'Project 1'

Start date: Wed 3/30/05

Finish date: Wed 3/30/05

Schedule from: Project Start Date

All tasks begin as soon as possible.

Current date: Wed 3/30/05

Status date: NA

Calendar: Standard

Priority: 500

Buttons: Help, Statistics..., OK, Cancel

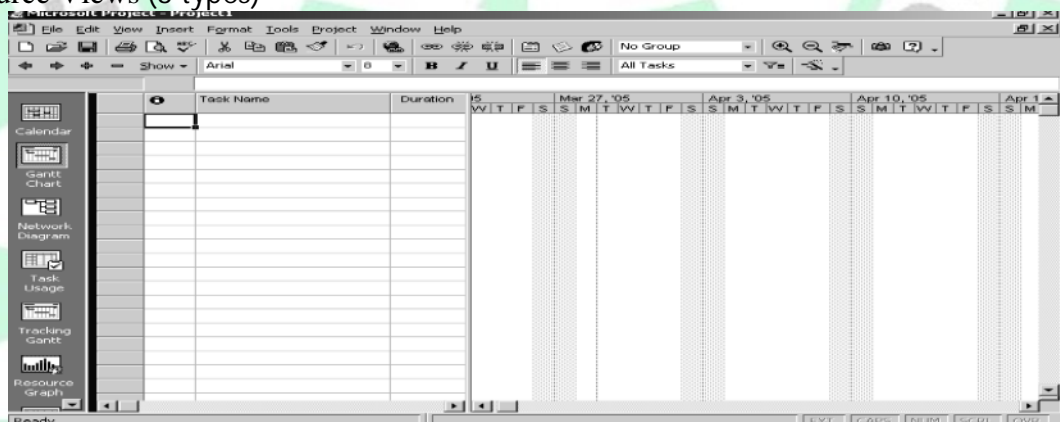
Enter the start date or select an appropriate date by scrolling down the list. Click OK. Project automatically enters a default start time and stores it as part of the dates entered and the application window is displayed.

Views

Views allow you to examine your project from different angles based on what information you want displayed at any given time. You can use a combination of views in the same window at the same time.

Project Views are categorized into two types:

- Task Views (5 types)
- Resource Views (3 types)



The Project worksheet is located in the main part of the application window and displays different information depending on the view you choose. The default view is the Gantt Chart View.

Goals of any project need to be defined in terms of tasks. There are four major types of tasks:

1. Summary tasks - contain subtasks and their related properties
2. Subtasks - are smaller tasks that are a part of a summary task
3. Recurring tasks - are tasks that occur at regular intervals
4. Milestones - are tasks that are set to zero duration and are like interim goals in the project

LINKS

Tasks are usually scheduled to start as soon as possible i.e. the first working day after the project start date.

Dependencies

Dependencies specify the manner in which two tasks are linked. Because Microsoft Project must maintain the manner in which tasks are linked, dependencies can affect the way a task is scheduled. In a scenario where there are two tasks, the following dependencies exist:

Finish to Start – Task 2 cannot start until task 1 finishes.

Start to Finish – Task 2 cannot finish until task 1 starts.

Start to Start – Task 2 cannot start until task 1 starts.

Finish to Finish – Task 2 cannot finish until task 1 finishes.

CONSTRAINTS

Certain tasks need to be completed within a certain date. Intermediate deadlines may need to be specified. By assigning constraints to a task you can account for scheduling problems.

To apply a constraint:

1. Open the Task Information dialog box.
2. Click the Advanced tab and open the Constraint type list by clicking on the drop-down arrow and select it.
3. Select a date for the Constraint and click OK.

RESOURCES

Once you determine that you need to include resources into your project you will need to answer the following questions:

- What kind of resources do you need?
- How many of each resource do you need?
- Where will you get these resources?
- How do you determine what your project is going to cost?

Resources are of two types - work resources and material resources.

Work resources complete tasks by expending time on them. They are usually people and equipment that have been assigned to work on the project.

Material resources are supplies and stocks that are needed to complete a project.

A new feature in Microsoft Project 2000 is that it allows you to track material resources and assign them to tasks

To assign a resource to a task:

1. Open the Gantt Chart View and the Assign Resources Dialog box.
2. In the Entry Table select the tasks for which you want to assign resources.
3. In the Assign Resources Dialog box, select the resource (resources) you want to assign.
4. Click the Assign button.

Case Study

MS Project to create a plan and time schedule for your term project. Use MS Project 2002 to create a series of tasks leading to completion of a project of your choice.

For your project, you need to:

Set start or ending dates.

Develop a list of tasks that need to be completed.

Establish any sub tasks and create links.

Create any links between major tasks.

Assign a specific amount time for each task.

Assign resources for each task.

Create task information for each item you put into the list.

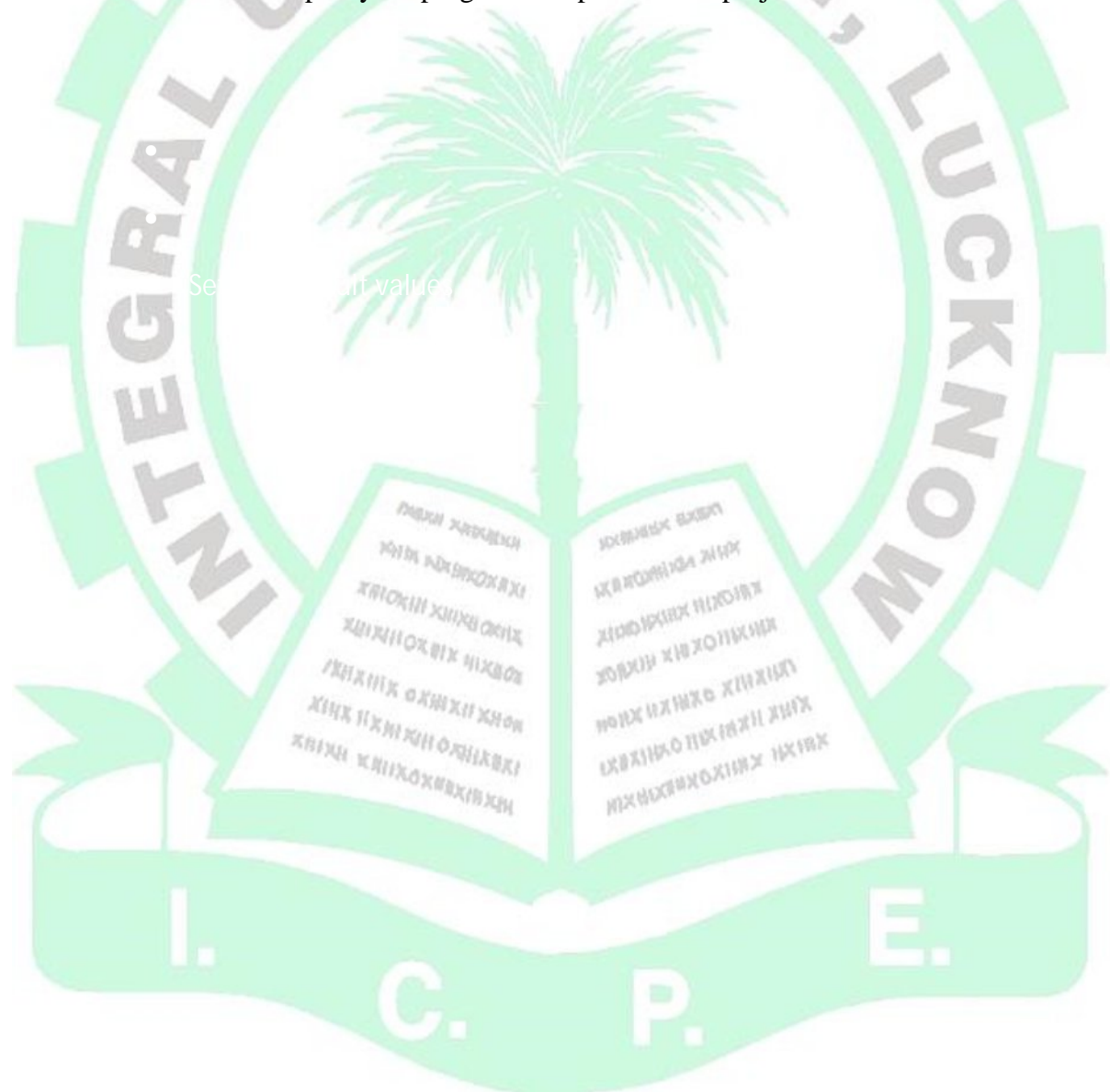
Project background

Palmers Golf Course is a two-year-old signature golf course based in Auckland. Being unique, the number of memberships is increasing and putting strain on the existing information system.

The current system is used only by Accounts Department to process Accounts Payables and Receivables. With the steep increase in the memberships, Bookings and Maintenance the Human Resource department is getting overwhelmed with the paper work. Therefore, there is a need to add these functions to a new system.

You are the Project Leader for Soft Systems Ltd., which has been contracted by Palmers Golf Course to undertake this project. You need to advise how to develop the new system in a tightly constrained time period. Specifically, you have been advised that the project can start no earlier than 2th January 2014 and must be completed by 22nd April 2014.

You report to Ms Smith, the Project Manager for SoftSystems Ltd, who liaises with Palmers Golf Course. You will report your progress and plans of the project to Ms Smith.



Experiment - 3

Objectives:

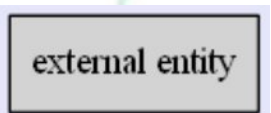


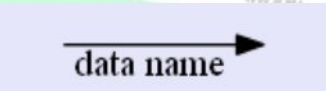
- Modeling Data Flow Diagrams
- Identify external entities and functionalities of any system
- Identify the flow of data across the system
- Represent the flow with Data Flow Diagrams

Theory

Data Flow Diagram

DFD provides the functional overview of a system. The graphical representation easily overcomes any gap between 'user and system analyst' and 'analyst and system designer' in understanding a system. Starting from an overview of the system it explores detailed design of a system through a hierarchy. DFD shows the external entities from which data flows into the process and also the other flows of data within a system. It also includes the transformations of data flow by the process and the data stores to read or write a data.

Graphical notations for Data Flow Diagram

Term	Notation	Remarks
External entity		Name of the external entity is written inside the rectangle
Process		Name of the process is written inside the circle
Data store		A left-right open rectangle is denoted as data store; name of the data store is written inside the shape
Data flow		Data flow is represented by a directed arc with its data name

Explanation of Symbols used in DFD

Process: Processes are represented by circle. The name of the process is written into the circle. The name of the process is usually given in such a way that represents the functionality of the process. More detailed functionalities can be shown in the next Level if it is required. Usually it is better to keep the number of processes less than 7 [i]. If we see that the number of processes becomes more than 7 then we should combine some the processes to a single one to reduce the number of processes and further decompose it to the next level [2] .

External entity: External entities are only appearing in context diagram [2]. External entities are represented by a rectangle and the name of the external entity is written into the shape. These send data to be processed and again receive the processed data.

Data store: Data stores are represented by a left-right open rectangle. Name of the data store is written in between two horizontal lines of the open rectangle. Data stores are used as repositories from which data can be flown in or flown out to or from a process.

Data flow: Data flows are shown as a directed edge between two components of a Data Flow Diagram. Data can flow from external entity to process, data store to process, in between two processes and vice-versa.

Context diagram and leveling DFD

We start with a broad overview of a system represented in level 0 diagrams. It is known as context diagram of the system. The entire system is shown as single process and also the interactions of external entities with the system are represented in context diagram. Further we split the process in next levels into several numbers of processes to represent the detailed functionalities performed by the system. Data stores may appear in higher level DFDs.

Numbering of processes : If process 'p' in context diagram is split into 3 processes 'p1', 'p2' and 'p3' in next level then these are labeled as 0.1, 0.2 and 0.3 in level 1 respectively. Let the process 'p3' is again split into three processes 'p31', 'p32' and 'p33' in level 2, so, these are labeled as 0.3.1, 0.3.2 and 0.3.3 respectively and so on.

Balancing DFD: The data that flow into the process and the data that flow out to the process need to be match when the process is split into in the next level[2]. This is known as balancing DFD.

Note :

External entities only appear in context diagram[2] i.e, only at level 0.

Keep number of processes at each level less than 7[i].

Data flow is not possible in between two external entities and in between two data stores[i].

Data cannot flow from an External entity to a data store and vice-versa[i].

Case Study

1 : A Library Information System for Institute

The Institute has been recently setup to provide state-of-the-art research facilities in the field of Software Engineering. Apart from research scholars (students) and professors, it also includes quite a large number of employees who work on different projects undertaken by the institution.

As the size and capacity of the institute is increasing with the time, it has been proposed to develop a Library Information System (LIS) for the benefit of students and employees of the institute. LIS will enable the members to borrow a book (or return it) with ease while sitting at his desk/chamber. The system also enables a member to extend the date of his borrowing if no other booking for that particular book has been made. For the library staff, this system aids them to easily handle day-to-day book transactions. The librarian, who has administrative privileges and complete control over the system, can enter a new record into the system when a new book has been purchased, or remove a record in case any book is taken off the shelf. Any non-member is free to use this system to browse/search books online. However, issuing or returning books is restricted to valid users (members) of LIS only.

The final deliverable would a web application, which should run only within the institute LAN. Although this reduces security risk of the software to a large extent, care should be taken no confidential information (eg., passwords) is stored in plain text.

Figure 1 shows the context-level DFD for LIS. The entire system is represented with a single circle (process). The external entities interacting with this system are members of LIS, library staff, librarian, and non-members of LIS. Two database are used to keep track of member information and details of books in the library.

Let us focus on the external entity, Member. In order to issue or return books a member has to login to the system. The data flow labeled with "Login credentials" indicate the step when a member authenticates himself by providing required information (user ID, password). The system in turn verifies the user credentials using information stored in the members database. If all information are not provided correctly, the user is shown a login failure message. Otherwise, the user can continue with his operation. Note that a DFD does not show conditional flows. It can only summarize the information flowing in and out of the system. The data flow with the label "Requested book details" identify the information that the user has to provide in order to issue a book. LIS checks with the books database whether the given book is available. After a book has been issued, the transaction details is provided to the member.

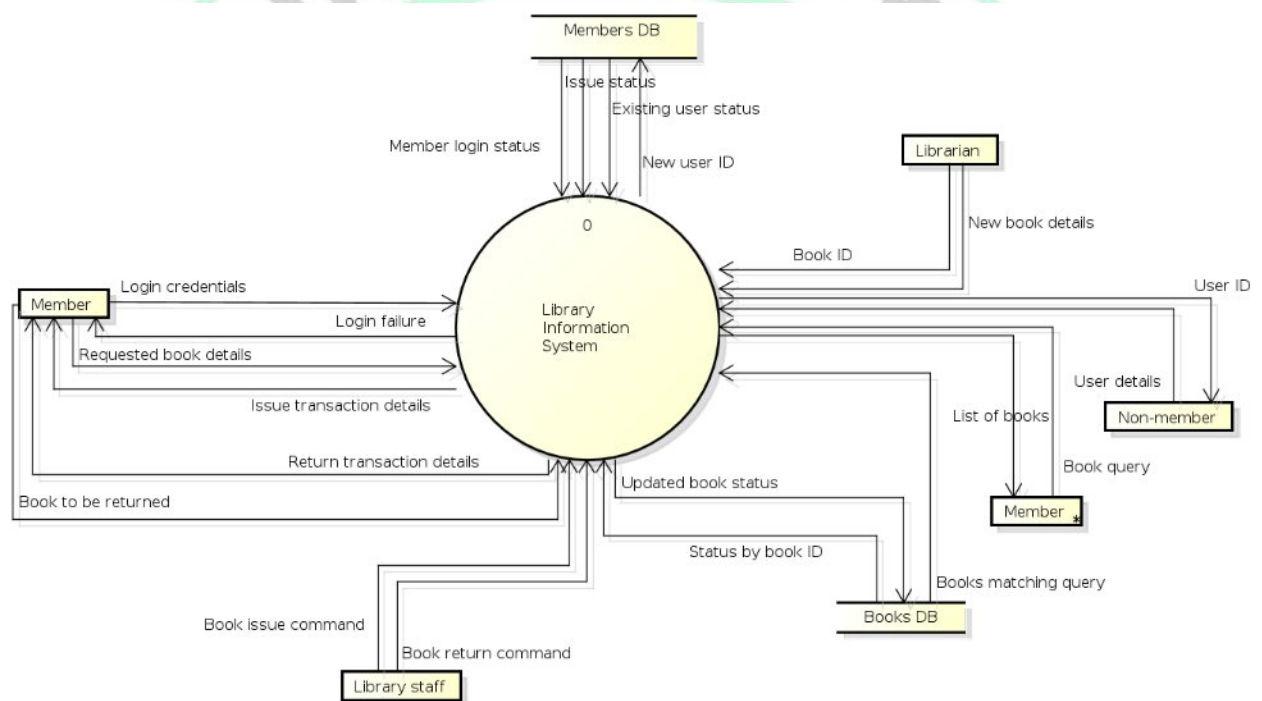


Figure 1: Context-level DFD for Library Information System

The level-1 DFD is shown in figure 2. Here, we split the top-level view of the system into multiple logical components. Each process has a name, and a dotted-decimal number in the form 1.x. For example, the process "Issue book" has the number 1.2, which indicates that in the level 1 DFD the concerned process is numbered 2. Other processes are numbered in a similar way.

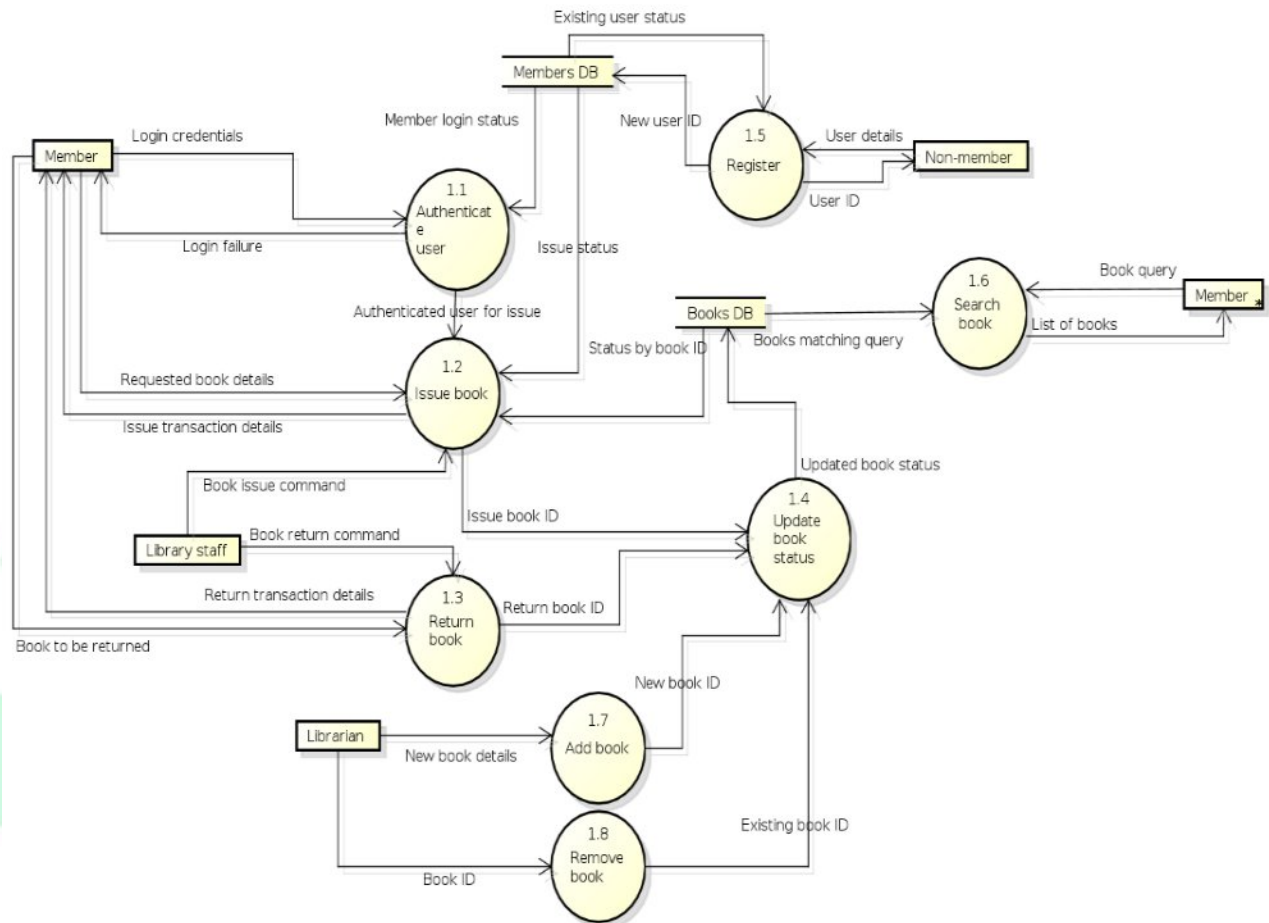


Figure 2: Level 1 DFD for Library Information System

Comparing figures 1 and 2 one might observe that the information flow in and out of LIS has been preserved. We observe in figure 2 that the sub-processes themselves exchange information among themselves. These information flows would be, in turn, preserved if we decompose the system into a level 2 DFD.

Finally, in order to eliminate intersecting lines and make the DFD complex, the Member external entity has been duplicated in figure 2. This is indicated by a * mark near the right-bottom corner of the entity box.

Exercise:

Design a DFD for AMT transaction.

DFD for a Social Networking site

The Absolute Beginners Inc. is planning to launch a revolutionary social networking site, EyeCopy. You have been entrusted with designing a DFD for the proposed application. In particular, you have been asked to show the following scenarios:

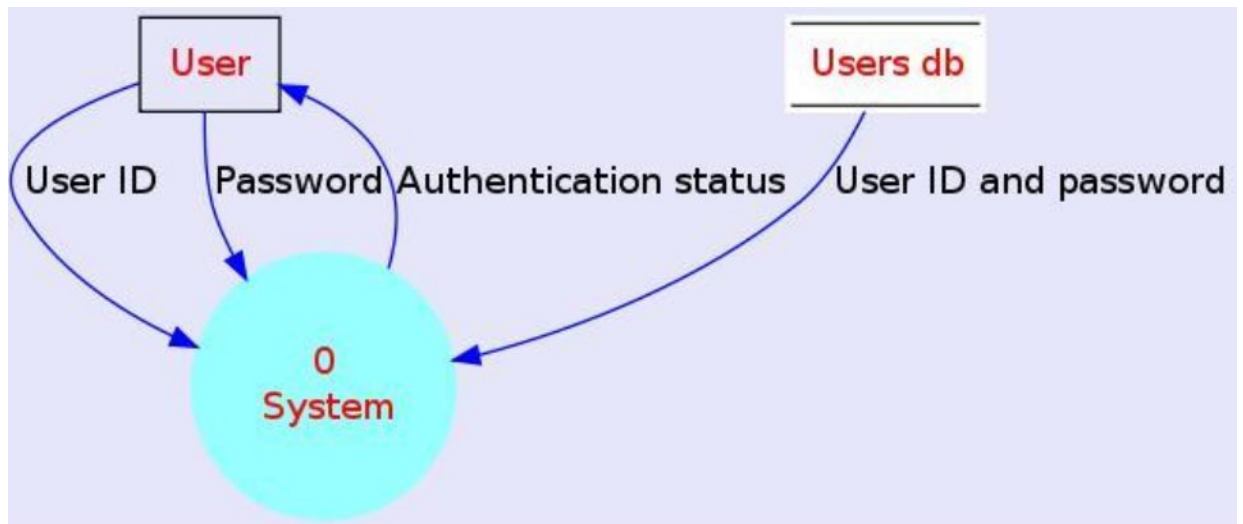
User registration

User login

Profile update

Draw a Level 1 DFD to depict the above data flow and the corresponding processes. Should there be any data store in the DFD?

Solution:



Bibliography

NPTEL course material - System Analysis and Design

Software Engineering Virtual Lab - Simulation

Software Engineering Virtual Lab - Case Study



Experiment - 4

Objectives:

- E-R Modeling from the Problem Statements.
- Identify entity sets, their attributes, and various relationships.
- Represent the data model through ER diagram.

Entity Relationship Model

Entity-Relationship model is used to represent a logical design of a database to be created. In ER model, real world objects (or concepts) are abstracted as entities, and different possible associations among them are modeled as relationships.

For example, student and school -- they are two entities. Students study in school. So, these two entities are associated with a relationship "Studies in".

As another example, consider a system where some job runs every night, which updates the database. Here, job and database could be two entities. They are associated with the relationship "Updates".

Entity Set and Relationship Set

An entity set is a collection of all similar entities. For example, "Student" is an entity set that abstracts all students. Ram, John are specific entities belonging to this set. Similarly, a "Relationship" set is a set of similar relationships.

Attributes of Entity

Attributes are the characteristics describing any entity belonging to an entity set. Any entity in a set can be described by zero or more attributes.

For example, any student has got a name, age, an address. At any given time a student can study only at one school. In the school he would have a roll number, and of course a grade in which he studies. These data are the attributes of the entity set Student.

Keys

One or more attribute(s) of an entity set can be used to define the following keys:

Super key: One or more attributes, which when taken together, helps to uniquely identify an entity in an entity set. For example, a school can have any number of students. However, if we know grade and roll number, then we can uniquely identify a student in that school.

Candidate key: It is a minimal subset of a super key. In other words, a super key might contain extraneous attributes, which do not help in identifying an object uniquely. When such attributes are removed, the key formed so is called a candidate key.

Primary key: A database might have more than one candidate key. Any candidate key chosen for a particular implementation of the database is called a primary key.

Prime attribute: Any attribute taking part in a super key

Weak Entity

An entity set is said to be weak if it is dependent upon another entity set. A weak entity can't be uniquely identified only by its attributes. In other words, it doesn't have a super key.

For example, consider a company that allows employees to have travel allowance for their immediate family. So, here we have two entity sets: employee and family, related by "Can claim for". However, family doesn't have a super key. Existence of a family is entirely dependent on the concerned employee. So, it is meaningful only with reference to employee.

Entity Generalization and Specialization

Once we have identified the entity sets, we might find some similarities among them. For example, multiple person interacts with a banking system. Most of them are customers, and rest employees or other service providers. Here, customers, employees are persons, but with

certain specializations. Or in other way, person is the generalized form of customer and employee entity sets.

ER model uses the "ISA" hierarchy to depict specialization (and thus, generalization).

Mapping Cardinalities

One of the main tasks of ER modeling is to associate different entity sets. Let's consider two entity sets E1 and E2 associated by a relationship set R. Based on the number of entities in E1 and E2 are associated with, we can have the following four type of mappings:

One to one: An entity in E1 is related to at most a single entity in E2, and vice versa

One to many: An entity in E1 could be related to zero or more entities in E2. Any entity in E2 could be related to at most a single entity in E1.



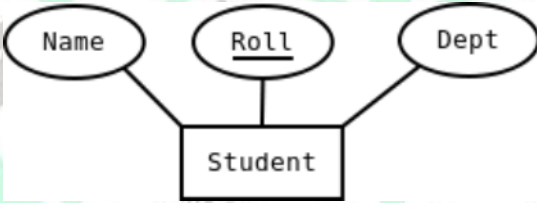
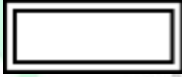
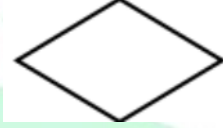


Many to one: Zero or more number of entities in E1 could be associated to a single entity in E2. However, an entity in E2 could be related to at most one entity in E1.


Many to many: Any number of entities could be related to any number of entities in E2, including zero, and vice versa.

ER Diagram

From a given problem statement we identify the possible entity sets, their attributes, and relationships among different entity sets. Once we have these information, we represent them pictorially, called an entity-relationship (ER) diagram.

Graphical Notations for ER Diagram

Term	Notation	Remarks
Entity set		Name of the set is written inside the rectangle
Attribute		Name of the attribute is written inside the ellipse
Entity with attributes		Roll is the primary key; denoted with an underline
Weak entity set		
Relationship set		Name of the relationship is written inside the diamond
Related entity sets		
Relationship cardinality		A person can own zero or more cars but no two persons can own the same car

Term	Notation	Remarks
Relationship with weak entity set		

Importance of ER modeling

Figure - 01 shows the different steps involved in implementation of a (relational) database.



Figure - 01: Steps to implement a RDBMS

Given a problem statement, the first step is to identify the entities, attributes and relationships. We represent them using an ER diagram. Using this ER diagram, table structures are created, along with required constraints. Finally, these tables are normalized in order to remove redundancy and maintain data integrity. Thus, to have data stored efficiently, the ER diagram is to be drawn as much detailed and accurate as possible.

Case Study

1 : A Library Information System for Institute

The SE VLabs Institute has been recently setup to provide state-of-the-art research facilities in the field of Software Engineering. Apart from research scholars (students) and professors, it also includes quite a large number of employees who work on different projects undertaken by the institution.

As the size and capacity of the institute is increasing with the time, it has been proposed to develop a Library Information System (LIS) for the benefit of students and employees of the institute. LIS will enable the members to borrow a book (or return it) with ease while sitting at his desk/chamber. The system also enables a member to extend the date of his borrowing if no other booking for that particular book has been made. For the library staff, this system aids them to easily handle day-to-day book transactions. The librarian, who has administrative privileges and complete control over the system, can enter a new record into the system when a new book has been purchased, or remove a record in case any book is taken off the shelf. Any non-member is free to use this system to browse/search books online. However, issuing or returning books is restricted to valid users (members) of LIS only.

The final deliverable would be a web application (using the recent HTML 5), which should run only within the institute LAN. Although this reduces security risk of the software to a large extent, care should be taken no confidential information (eg., passwords) is stored in plain text.

A robust database backend is essential for a high-quality information system. Database schema should be efficiently modeled, refined, and normalized. In this section we would develop a simple ER model for the Library Information System.

The first step towards ER modeling is to identify the set of relevant entities from the given problem statement. The two primary, and obvious, entity sets in this context are "Member" and "Book". The entity set "Member" represents all students, professors, or employees who have registered themselves with the LIS. While registering with the LIS one has to furnish a lot of personal and professional information. This typically includes name (well, that is trivial), employee ID (roll # for students), email address, phone #, age, date of joining in this institute. The system may store some not-so-important information as well like, blood group, marital status, and so on. All these pieces of information that an user has to provide are sufficient to describe a particular member. These characteristics are the attributes of the entities belonging to the entity set "Member".

It is essential for an entity to have one or more attributes that help us to distinguish it from another entity. 'Name' can't help that -- two persons could have exactly the same name. However, ('Name', 'Phone #') combination seems to be okay. No two persons can have the same phone number. 'Employee ID', 'Email address' are other potential candidates. Here, 'Employee ID', 'Email address' and ('Name', 'Phone #') are super keys. We choose 'Employee ID' to uniquely identify an user in our implementation. So, 'Employee ID' becomes our primary key (PK) for the "Member" entity set. Figure 1 represents this set along with its attributes and the primary key.

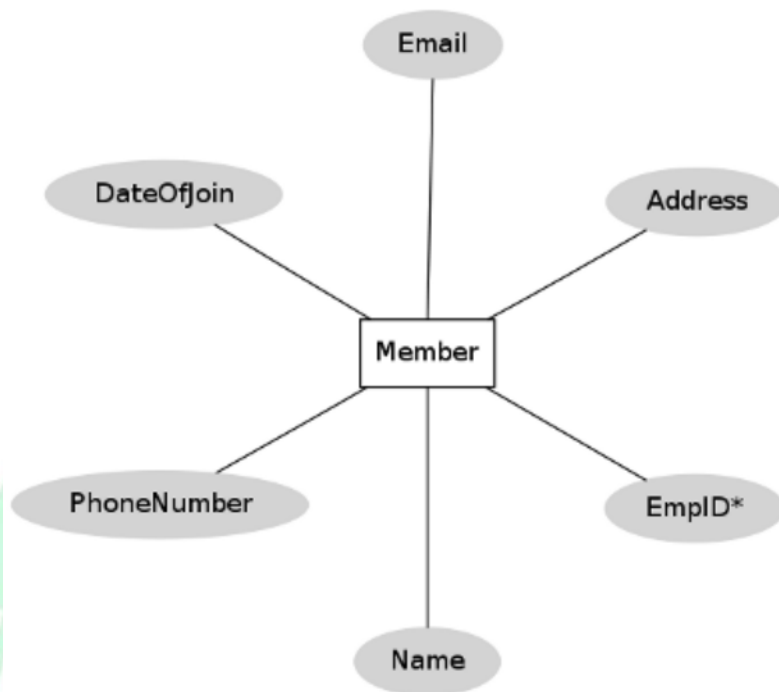


Figure 1: "Member" entity set

Let us now focus on the "Book" entity set. Typical attributes of a book are its title, name of author(s), publisher, date of publication, edition, language, ISBN-10, ISBN-13, price (of course!), date of purchase. The set of listed attributes for a book doesn't give a straight forward choice of primary key. For instance, several books could have the same title. Again, ISBN numbers for a book are specific to its edition -- it can't distinguish between two books of the same edition. One might be tempted to use a combination of ('Title', 'Authors') as a primary key. This has some shortcomings. It is advisable not to use texts as a PK. Moreover, the number of authors that a book could have is not fixed, although it is a small, finite number. The rules of normalization (not covered here) would dictate to have a separate field for each author like 'Author1', 'Author2', and so on. Therefore, we assign an extra attribute, 'ID', to each book as its PK. Different databases available in the market provide mechanisms to generate such a unique ID, and automatically increment it whenever a new new entity is added. In fact, we could assign such an ID to the "Member" entity set as well. However, because of availability of the unique 'Employee ID' field, we skipped that. A graphical representation of the "Book" entity set is shown in figure 2.

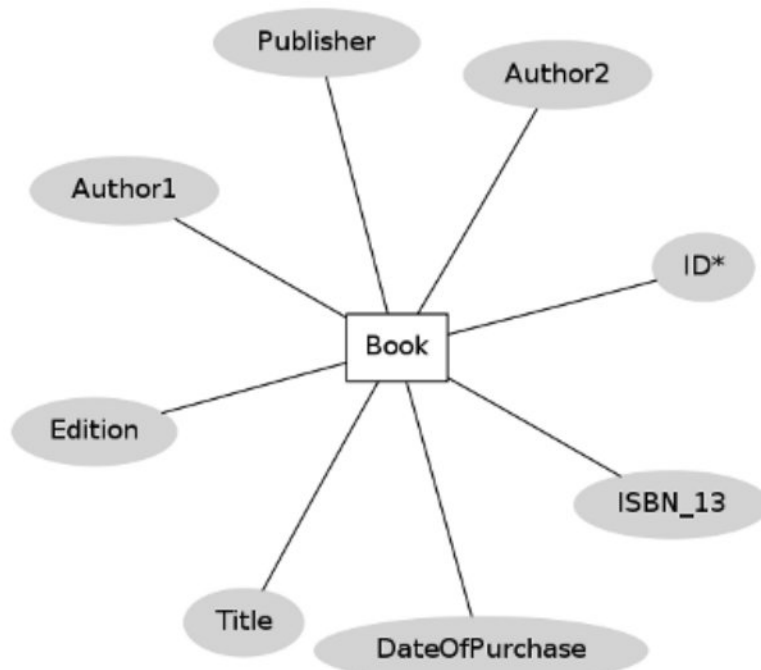


Figure 2: "Book" entity set

One point to note here is that a book is likely to have multiple copies in the library. Therefore, one might wish to have a '# of copies' attribute for the "Book" entity set. However, that won't allow us to differentiate among the different copies of book bearing same title by same author(s), edition, and publisher. The approach that we have taken is to uniquely identify each book even though they are copies of the same title.

To buy any new book an order is to be placed to the distributor. This task is done by the librarian. Therefore, "Librarian" and "Distributor" are two other entities playing roles in this system.

Having identified the key entities, we could now relate them with each other. Let us consider the entity sets "Member" and "Book". A member can issue books. In fact, he can issue multiple books up to a finite number say, N. A particular book, however, could be issued by a single member only. Therefore, we have a one-to-many mapping from "Member" to "Book" entity sets. This relationship between "Member" and "Book" entity sets is pictorially depicted in figure 3.

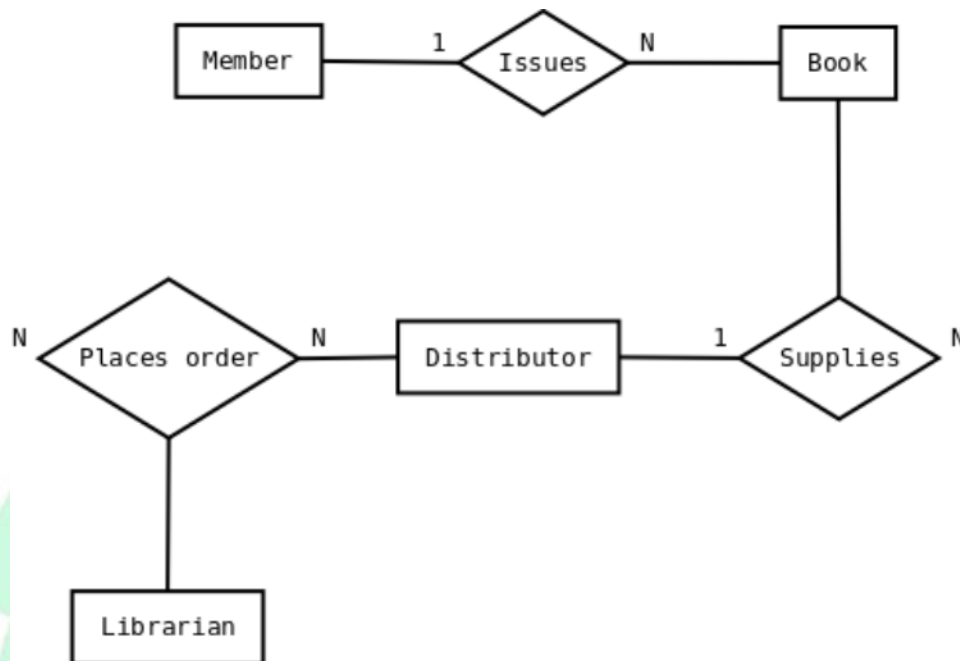


Figure 3: Relationships among different entity sets

Figure 3 also shows that the librarian can "place order" for books to the distributor. This is a many-to-many mapping since a librarian can purchase books from multiple distributors. Also, if the institute has more than one librarians (or any other staff having such authority), then each of them could place order to the same distributor. An order is termed as complete when distributor supplies the book(s) and invoice.

The design in figure 3 has a flaw. Librarian himself could be a member of the LIS. However, he is a "special" kind of member since he can place order for books. Our ER diagram doesn't reflect this scenario. Such special roles of an entity set could be represented using "ISA" relationship, which is not discussed here.

Any kind of designing couldn't be possibly done at one go. Therefore, the baseline ER model so prepared should be revised by considering the business model yet again to ensure that all necessary information could be captured. Once this has been finalized, the next logical step would be to create table structures for each identified entity set (and relationships in some cases) and normalize the relations.

Exercise:

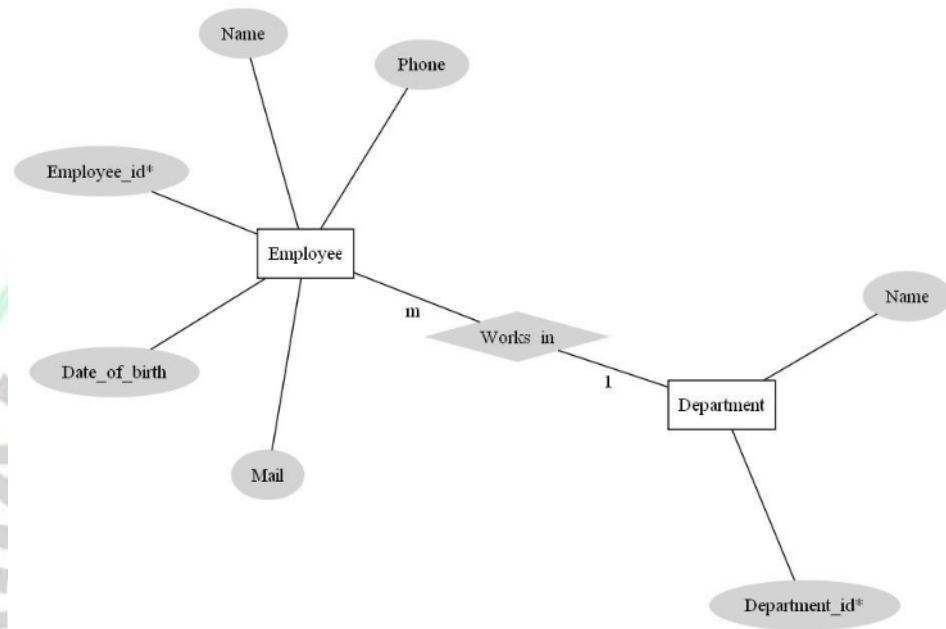
1. From the following problem statement identify the possible entity sets, their attributes and relationships.

SE VLabs Inc. is a young company with a few departments spread across the country. As of now, the company has a strength of 200+ employees.

Each employee works in a department. While joining, a person has to provide a lot of personal and professional details including name, address, phone #, mail address, date of birth, and so on. Once all these information are furnished, a unique ID is generated for each employee. He is then assigned a department in which he will work.

There are around ten departments in the company. Unfortunately, two departments were given same names. However, departments too have ID's, which are unique.

Solution:



Draw an ER diagram for the following problem:

The latest cab services agency in the city has approached you to develop a Cab Management System for them. They would be using this software to efficiently manage and track different cabs that are operated by them.

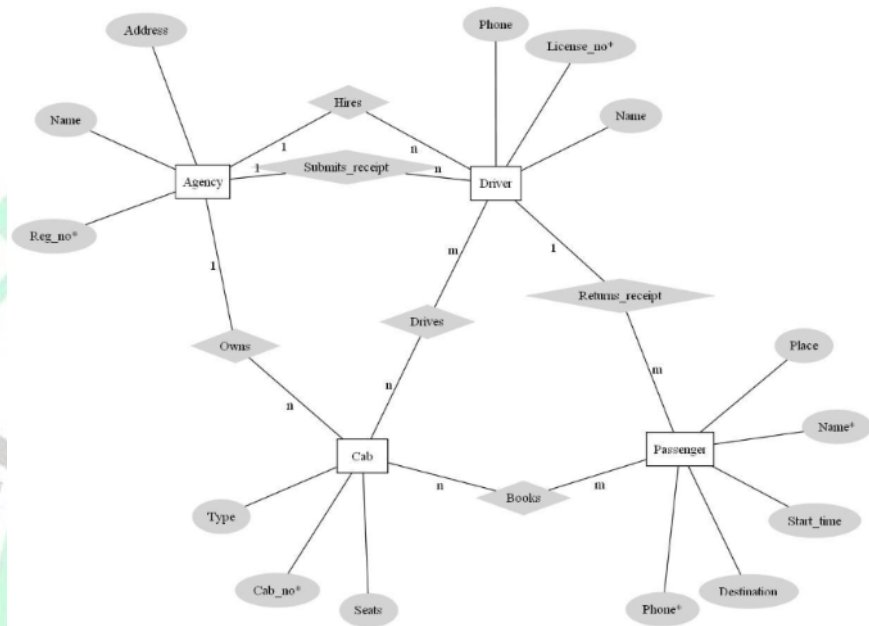
Cabs are solely owned by the agency. They hire people in contracts to drive the cabs. A cab can be uniquely identified by, like any other vehicle in the country, its license plate. A few different categories of cars are available from different manufacturers. And a few of them are AC cars.

Cab drivers are given a identification card while joining. The ID card contains his name, permanent address, phone number, date of joining, duration of contract. Also, an unique alphanumeric code is assigned to each number.

The agency provides service from 8 AM to 8 PM. Whenever any passenger books a cab, an available cab is allocated for him. The booking receipt given to the passenger contains the car #, source and destination places. Once he reaches the destination, he signs on a duplicate copy of the receipt and gives back to the driver. Driver must submit this duplicate copy signed by the passenger at the agency for confirmation.

To evaluate their quality of service, the agency also wants a (optional) customer satisfaction survey, where passengers can provide feedback about their journey through the agency's website.

Solution:



Bibliography

Database System Concepts, Henry F. Korth , Abraham Silberschatz, McGraw Hills, 5th Edition

Entity-relationship modeling

Entity Relationship Modelling - 2

Experiment -5

Objective: General study of UML

- What is UML
- Why the UML is necessary
- How to represent UML components in diagrams

Theory: The heart of object-oriented problem solving is the construction of a model. The model abstracts the essential details of the underlying problem from its usually complicated real world. Several modeling tools are wrapped under the heading of the UML™, which stands for Unified Modeling Language™. The purpose of this course is to present important highlights of the UML.

At the center of the UML are its nine kinds of modeling diagrams, which we describe here.

- Use case diagrams
- Class diagrams
- Object diagrams
- Sequence diagrams
- Collaboration diagrams
- Statechart diagrams
- Activity diagrams
- Component diagrams
- Deployment diagrams

Some of the sections of this course contain links to pages with more detailed information. And every section has short questions. Use them to test your understanding of the section topic.

Why is UML important?

Let's look at this question from the point of view of the construction trade. Architects design buildings. Builders use the designs to create buildings. The more complicated the building, the more critical the communication between architect and builder. Blueprints are the standard graphical language that both architects and builders must learn as part of their trade.

Writing software is not unlike constructing a building. The more complicated the underlying system, the more critical the communication among everyone involved in creating and deploying the software. In the past decade, the UML has emerged as the software blueprint language for analysts, designers, and programmers alike. It is now part of the software trade. The UML gives everyone from business analyst to designer to programmer a common vocabulary to talk about software design.

The UML is applicable to object-oriented problem solving. Anyone interested in learning UML must be familiar with the underlying tenet of object-oriented problem solving -- it all begins with the construction of a model. A model is an abstraction of the underlying problem. The domain is the actual world from which the problem comes. Models consist of objects that interact by sending each other messages. Think of an object as "alive." Objects have things they know (attributes) and things they can do (behaviors or operations). The values of an object's attributes determine its state.

Classes are the "blueprints" for objects. A class wraps attributes (data) and behaviors (methods or functions) into a single distinct entity. Objects are instances of classes.

Use case diagrams

Use case diagrams describe what a system does from the standpoint of an external observer. The emphasis is on what a system does rather than how.

Use case diagrams are closely connected to scenarios. A scenario is an example of what happens when someone interacts with the system. Here is a scenario for a medical clinic.

"A patient calls the clinic to make an appointment for a yearly checkup. The receptionist finds the nearest empty time slot in the appointment book and schedules the appointment for that time slot."

A use case is a summary of scenarios for a single task or goal. An actor is who or what initiates the events involved in that task. Actors are simply roles that people or objects play. The picture below is a Make Appointment use case for the medical clinic. The actor is a Patient. The connection between actor and use case is a communication association (or communication for short).

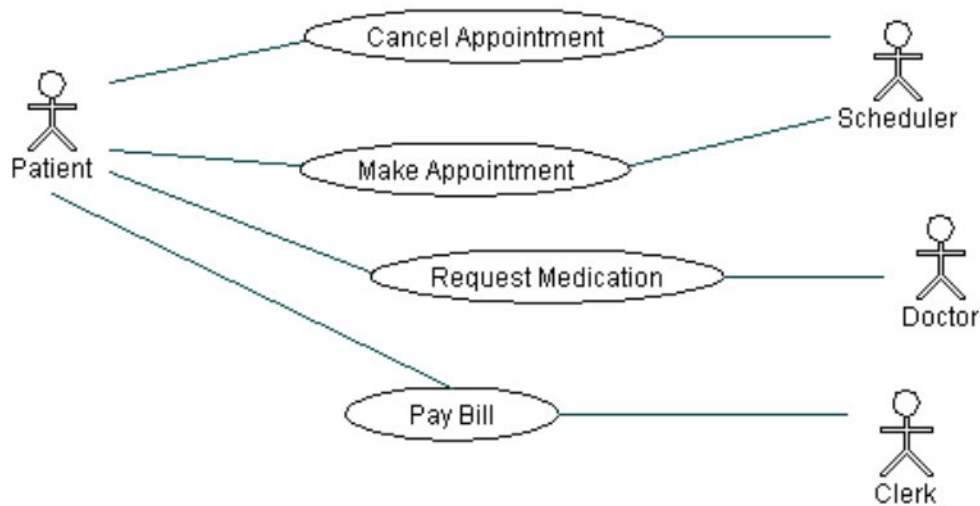


Actors are stick figures. Use cases are ovals. Communications are lines that link actors to use cases.

A use case diagram is a collection of actors, use cases, and their communications. We've put Make Appointment as part of a diagram with four actors and four use cases. Notice that a single use case can have multiple actors.

Use case diagrams are helpful in three areas.

- determining features (requirements). New use cases often generate new requirements as the system is analyzed and the design takes shape.
- communicating with clients. Their notational simplicity makes use case diagrams a good way for developers to communicate with clients.
- generating test cases. The collection of scenarios for a use case may suggest a suite of test cases for those scenarios.



Class diagrams

A Class diagram gives an overview of a system by showing its classes and the relationships among them. Class diagrams are static -- they display what interacts but not what happens when they do interact.

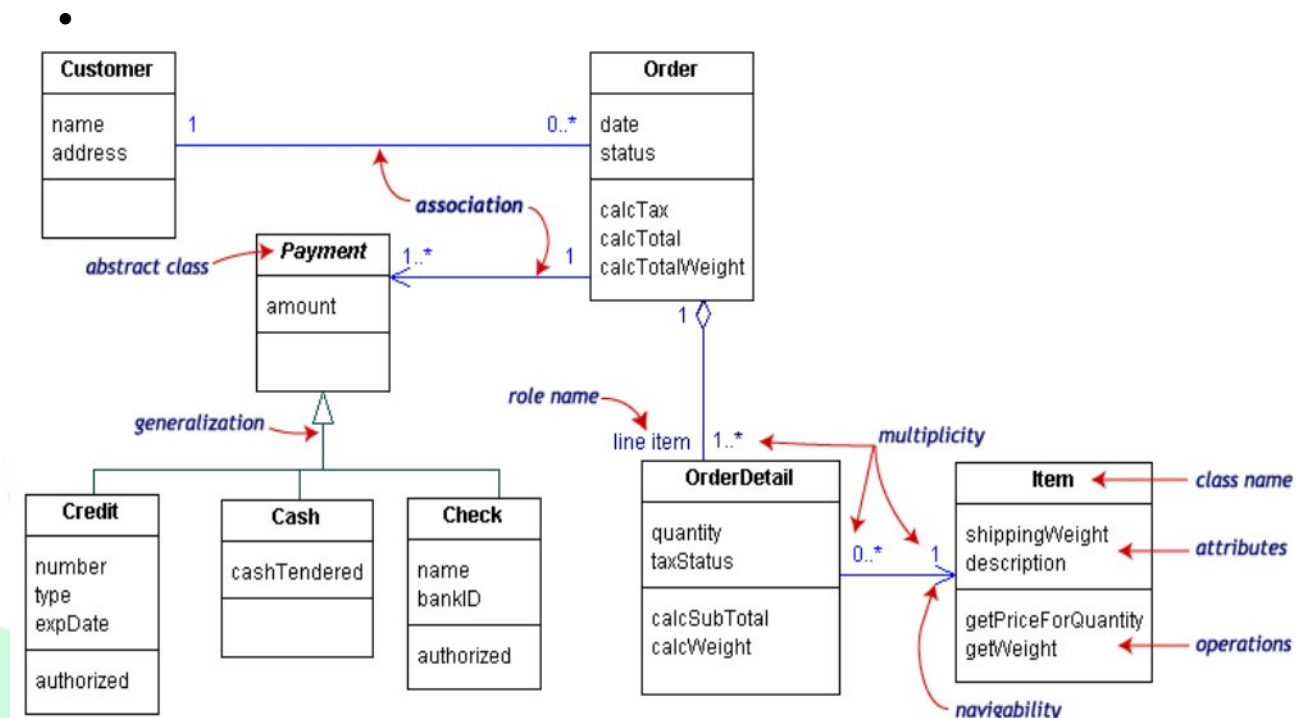
The class diagram below models a customer order from a retail catalog. The central class is the Order. Associated with it are the Customer making the purchase and the Payment. A Payment is one of three kinds: Cash, Check, or Credit. The order contains OrderDetails (line items), each with its associated Item.

UML class notation is a rectangle divided into three parts: class name, attributes, and operations. Names of abstract classes, such as Payment, are in italics. Relationships between classes are the connecting links.

Our class diagram has three kinds of relationships.

- **association** -- a relationship between instances of the two classes. There is an association between two classes if an instance of one class must know about the other in order to perform its work. In a diagram, an association is a link connecting two classes.
- **aggregation** -- an association in which one class belongs to a collection. An aggregation has a diamond end pointing to the part containing the whole. In our diagram, Order has a collection of OrderDetails.
- **generalization** -- an inheritance link indicating one class is a superclass of the other. A generalization has a triangle pointing to the superclass. Payment is a superclass of Cash, Check, and Credit.
- An association has two ends. An end may have a role name to clarify the nature of the association. For example, an OrderDetail is a line item of each Order.
- A navigability arrow on an association shows which direction the association can be traversed or queried. An OrderDetail can be queried about its Item, but not the other way around. The arrow also lets you know who "owns" the association's implementation; in this case, OrderDetail has an Item. Associations with no navigability arrows are bi-directional.
- The multiplicity of an association end is the number of possible instances of the class associated with a single instance of the other end. Multiplicities are single

numbers or ranges of numbers. In our example, there can be only one Customer for each Order, but a Customer can have any number of Orders.



This table gives the most common multiplicities.

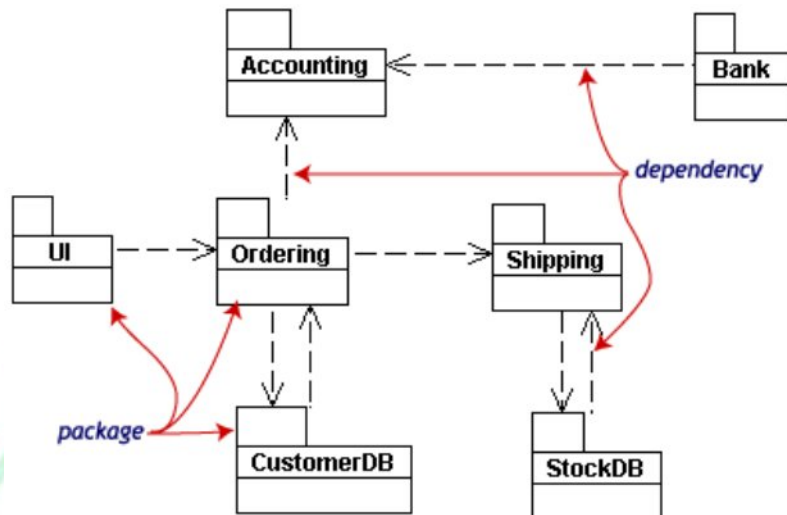
Multiplicities	Meaning
0..1	zero or one instance. The notation n . . m indicates n to m instances.
0..* or *	no limit on the number of instances (including none).
1	exactly one instance
1..*	at least one instance

Every class diagram has classes, associations, and multiplicities. Navigability and roles are optional items placed in a diagram to provide clarity.

Packages and object diagrams

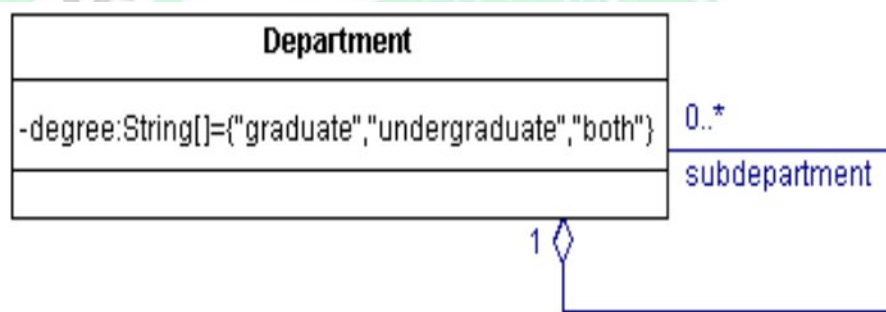
To simplify complex class diagrams, you can group classes into packages. A package is a collection of logically related UML elements. The diagram below is a business model in which the classes are grouped into packages.

Packages appear as rectangles with small tabs at the top. The package name is on the tab or inside the rectangle. The dotted arrows are dependencies. One package depends on another if changes in the other could possibly force changes in the first.

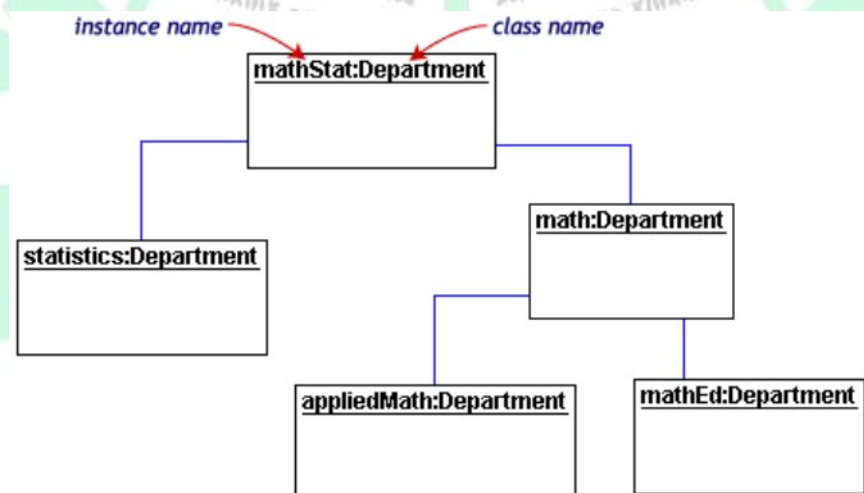


Object diagrams show instances instead of classes. They are useful for explaining small pieces with complicated relationships, especially recursive relationships.

This small class diagram shows that a university Department can contain lots of other Departments.



The object diagram below instantiates the class diagram, replacing it by a concrete example.



Each rectangle in the object diagram corresponds to a single instance. Instance names are underlined in UML diagrams. Class or instance names may be omitted from object diagrams as long as the diagram meaning is still clear.

Sequence diagrams

Class and object diagrams are static model views. Interaction diagrams are dynamic. They describe how objects collaborate.

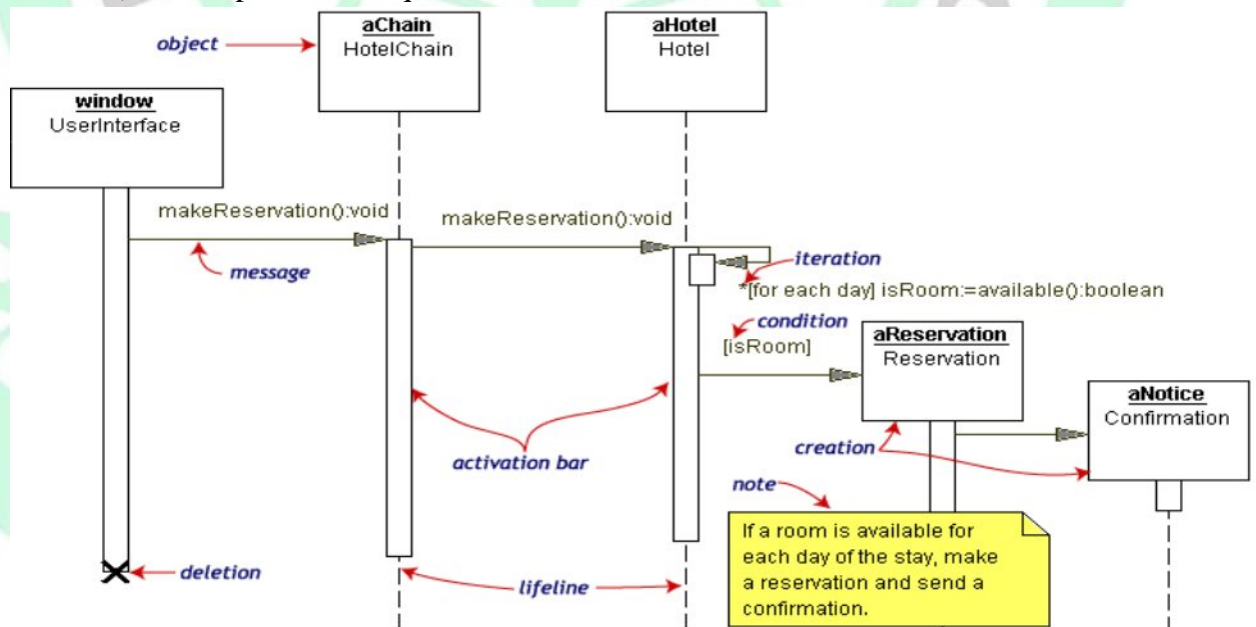
A sequence diagram is an interaction diagram that details how operations are carried out -- what messages are sent and when. Sequence diagrams are organized according to time. The time progresses as you go down the page. The objects involved in the operation are listed from left to right according to when they take part in the message sequence.

Below is a sequence diagram for making a hotel reservation. The object initiating the sequence of messages is a Reservation window.

The Reservation window sends a makeReservation() message to a HotelChain. The HotelChain then sends a makeReservation() message to a Hotel. If the Hotel has available rooms, then it makes a Reservation and a Confirmation.

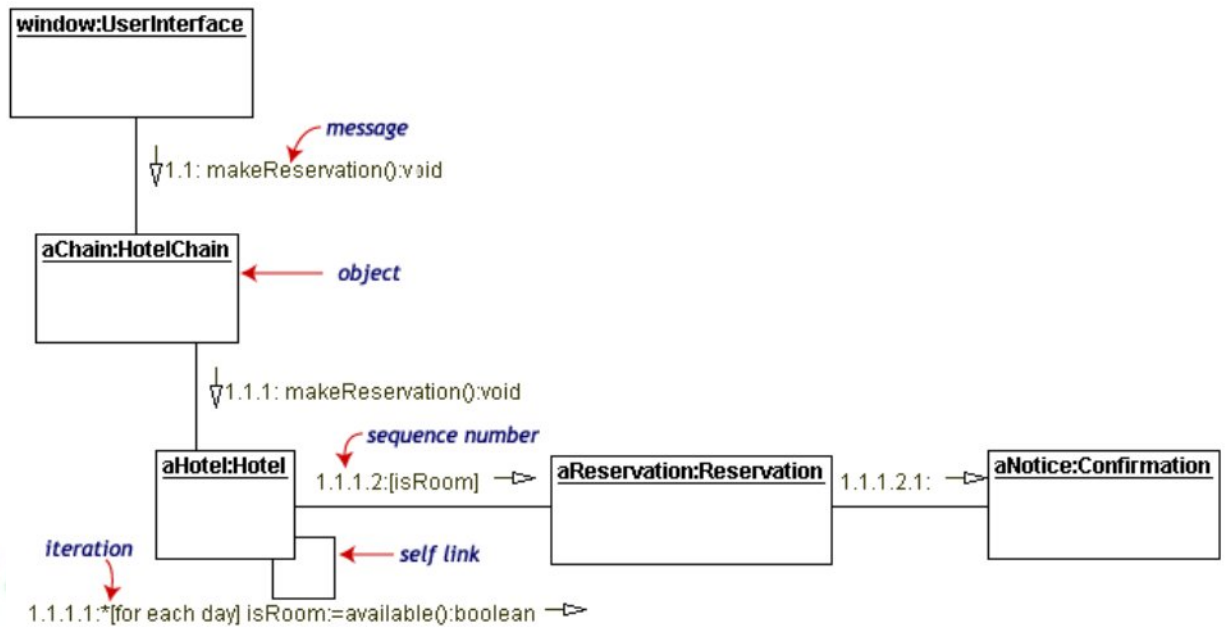
Each vertical dotted line is a lifeline, representing the time that an object exists. Each arrow is a message call. An arrow goes from the sender to the top of the activation bar of the message on the receiver's lifeline. The activation bar represents the duration of execution of the message.

In our diagram, the Hotel issues a self call to determine if a room is available. If so, then the Hotel creates a Reservation and a Confirmation. The asterisk on the self call means iteration (to make sure there is available room for each day of the stay in the hotel). The expression in square brackets, [], is a condition.



Collaboration diagrams

Collaboration diagrams are also interaction diagrams. They convey the same information as sequence diagrams, but they focus on object roles instead of the times that messages are sent. In a sequence diagram, object roles are the vertices and messages are the connecting links.



The object-role rectangles are labeled with either class or object names (or both). Class names are preceded by colons (:).

Each message in a collaboration diagram has a sequence number. The top-level message is numbered 1. Messages at the same level (sent during the same call) have the same decimal prefix but suffixes of 1, 2, etc. according to when they occur.

State chart diagrams

Objects have behaviors and state. The state of an object depends on its current activity or condition. A statechart diagram shows the possible states of the object and the transitions that cause a change in state.

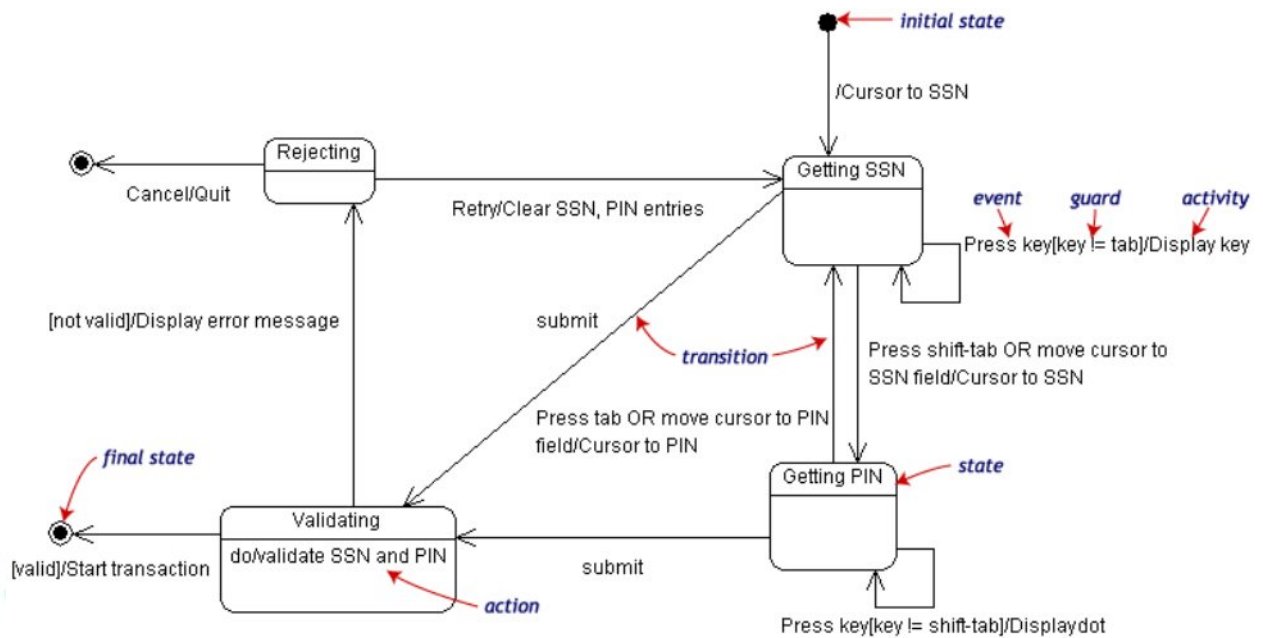
Our example diagram models the login part of an online banking system. Logging in consists of entering a valid social security number and personal id number, then submitting the information for validation.

Logging in can be factored into four non-overlapping states: Getting SSN, Getting PIN, Validating, and Rejecting. From each state comes a complete set of transitions that determine the subsequent state.

States are rounded rectangles. Transitions are arrows from one state to another. Events or conditions that trigger transitions are written beside the arrows. Our diagram has two self-transition, one on Getting SSN and another on Getting PIN.

The initial state (black circle) is a dummy to start the action. Final states are also dummy states that terminate the action.

The action that occurs as a result of an event or condition is expressed in the form /action. While in its Validating state, the object does not wait for an outside event to trigger a transition. Instead, it performs an activity. The result of that activity determines its subsequent state.



Activity diagrams

An activity diagram is essentially a fancy flowchart. Activity diagrams and statechart diagrams are related. While a statechart diagram focuses attention on an object undergoing a process (or on a process as an object), an activity diagram focuses on the flow of activities involved in a single process. The activity diagram shows how those activities depend on one another.

For our example, we used the following process.

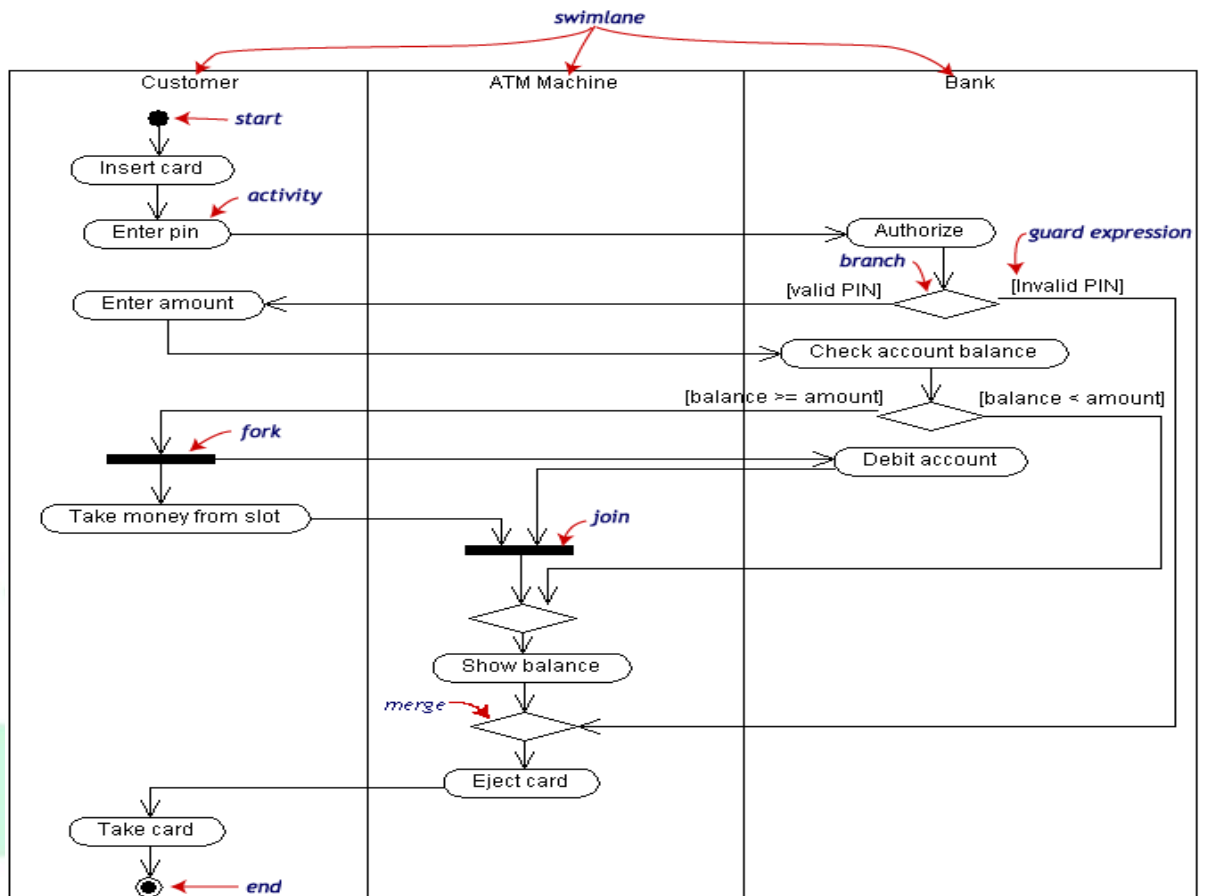
"Withdraw money from a bank account through an ATM."

The three involved classes (people, etc.) of the activity are Customer, ATM, and Bank. The process begins at the black start circle at the top and ends at the concentric white/black stop circles at the bottom. The activities are rounded rectangles.

Activity diagrams can be divided into object swimlanes that determine which object is responsible for which activity. A single transition comes out of each activity, connecting it to the next activity.

A transition may branch into two or more mutually exclusive transitions. Guard expressions (inside []) label the transitions coming out of a branch. A branch and its subsequent merge marking the end of the branch appear in the diagram as hollow diamonds.

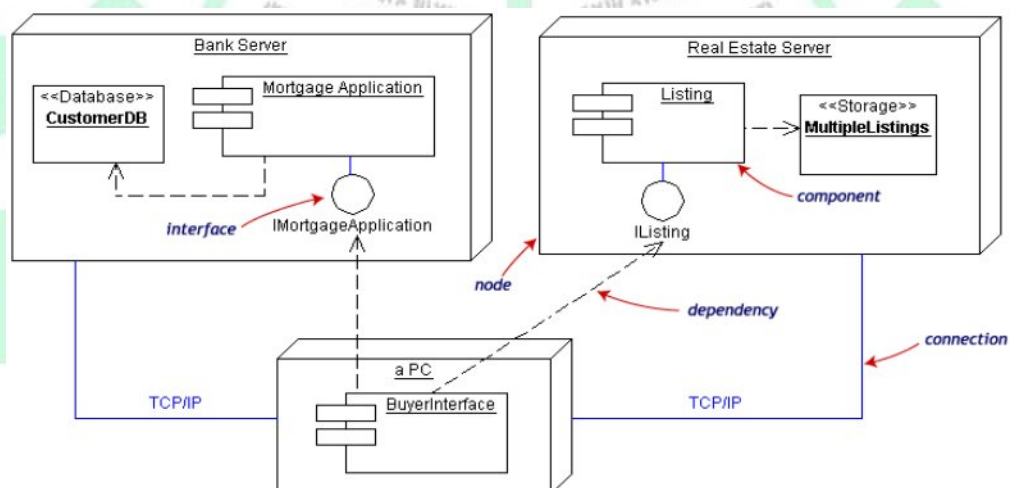
A transition may fork into two or more parallel activities. The fork and the subsequent join of the threads coming out of the fork appear in the diagram as solid bars.



Component and deployment diagrams

A component is a code module. Component diagrams are physical analogs of class diagram. Deployment diagrams show the physical configurations of software and hardware.

The following deployment diagram shows the relationships among software and hardware components involved in real estate transactions.



The physical hardware is made up of nodes. Each component belongs on a node. Components are shown as rectangles with two tabs at the upper left.

Exercise:

1. Design all UML diagrams given in experiment no 4 using Ms Visio/ IBM Rational Rose.
2. Study various tools of Ms Visio/ IBM Rational Rose.

Bibliography

<http://homepages.uel.ac.uk/D.Bowden/index.htm>

www.uml.org.cn/umltools/psf/UsingRationalRose.pdf



Experiment No 6

Objective: UML Use Case Diagram

- Study about use case diagram
- Design use case diagram for given problem

Modeling steps for Use case Diagram

1. Draw the lines around the system and actors lie outside the system.
2. Identify the actors which are interacting with the system.
3. Separate the generalized and specialized actors.
4. Identify the functionality the way of interacting actors with system and specify the behavior of actor.
5. Functionality or behavior of actors is considered as use cases.
6. Specify the generalized and specialized use cases.
7. See the relationship among the use cases and in between actor and use cases.
8. Adorn with constraints and notes.
9. If necessary, use collaborations to realize use cases.

UML Use Case Diagram

To model a system the most important aspect is to capture the dynamic behavior. To clarify a bit in details, dynamic behavior means the behavior of the system when it is running/operating.

So only static behavior is not sufficient to model a system rather dynamic behavior is more important than static behavior. In UML there are five diagrams available to model dynamic nature and use case diagram is one of them. Now as we have to discuss that the use case diagram is dynamic in nature there should be some internal or external factors for making the interaction.

These internal and external agents are known as actors. So use case diagrams consist of actors, use cases and their relationships. The diagram is used to model the system/subsystem of an application. A single use case diagram captures a particular functionality of a system. So to model the entire system numbers of use case diagrams are used.

Purpose:

The purpose of use case diagram is to capture the dynamic aspect of a system. But this definition is too generic to describe the purpose.

Because other four diagrams (activity, sequence, collaboration and Statechart) are also having the same purpose. So we will look into some specific purpose which will distinguish it from other four diagrams.

Use case diagrams are used to gather the requirements of a system including internal and external influences. These requirements are mostly design requirements. So when a system is analyzed to gather its functionalities use cases are prepared and actors are identified.

Now when the initial task is complete use case diagrams are modelled to present the outside view. So in brief, the purposes of use case diagrams can be as follows:

- Used to gather requirements of a system.
- Used to get an outside view of a system.
- Identify external and internal factors influencing the system.
- Show the interacting among the requirements are actors.

How to draw Use Case Diagram?

Use case diagrams are considered for high level requirement analysis of a system. So when the requirements of a system are analyzed the functionalities are captured in use cases.

So we can say that uses cases are nothing but the system functionalities written in an organized manner. Now the second things which are relevant to the use cases are the actors. Actors can be defined as something that interacts with the system.

The actors can be human user, some internal applications or may be some external applications. So in a brief when we are planning to draw an use case diagram we should have the following items identified.

- Functionalities to be represented as an use case
- Actors
- Relationships among the use cases and actors.

Use case diagrams are drawn to capture the functional requirements of a system. So after identifying the above items we have to follow the following guidelines to draw an efficient use case diagram.

- The name of a use case is very important. So the name should be chosen in such a way so that it can identify the functionalities performed.
- Give a suitable name for actors.
- Show relationships and dependencies clearly in the diagram.
- Do not try to include all types of relationships. Because the main purpose of the diagram is to identify requirements.
- Use note when ever required to clarify some important points.

The following is a sample use case diagram representing the order management system. So if we look into the diagram then we will find three use cases (Order, Special Order and Normal Order) and one actor which is customer.

The Special Order and Normal Order use cases are extended from Order use case. So they have extends relationship. Another important point is to identify the system boundary which is shown in the picture. The actor Customer lies outside the system as it is an external user of the system.

Case Study: Library System

1. Problem Statement

The Library System is a web-based application used to automate a library. It allows the librarian to maintain the information about books, magazines and CDs. It also allows the librarian to maintain the information about its users. It provides the facilities such as search for items, browse, checkout items, return items, make reservation, remove reservation etc. to its users.

To borrow the items from the library, the users must register in the system. The search option allows the users to search for any item in the library. If the user finds that the required item is available in the library, he/she can check out the item from the library. If the item is currently

not available in the library, the user can make reservation for the item. When the item becomes available the respective user who made the reservation for that item first is notified. The reservation is canceled when the user checks out the item from the library or through an explicit cancellation procedure.

The system allows the librarian to easily create, update, and delete information about titles, borrowers, items and reservations in the system. The librarian is an employee of the library who interacts with the borrowers whose work is supported by the system.

The Library System can run on popular web-browser platforms like Windows Explorer, Netscape Navigator etc. It can be easily extended with new functionality.

2. Vision Document

A vision document describes the higher level requirements of the system specifying the scope of the system.

The vision document for the Library System might be

- It is a support system
- The library lends books, magazines and CDs to borrowers who are registered in the system
- The Library System handles the purchases of new titles for the library
- Popular titles are brought in multiple copies. Old books, magazines and CDs are removed when they are out of date or in poor condition
- The librarian is an employee of the library who interacts with the borrowers whose work is supported by the system
- A borrower can reserve a book, magazine or CD that is not currently available in the library so that when it is returned or purchased by the library, the borrower is notified
- The reservation is canceled when the borrower checks out the book, magazine or CD or through an explicit cancellation procedure
- The librarian can easily create, update, and delete information about titles, borrowers, items and reservations in the system
- The system can run on popular web-browser platforms like Windows Explorer, Netscape navigator etc.
- The system is easy to extend with new functionality

3. Glossary

Key terms are denoted in *italics* within the use-case specifications.

Item - A tangible copy of a **Title**. **Title** - The descriptive identifying information for a book or magazine. Includes attributes like name and description. **Reservation** - Whenever a borrower wishes to checkout an **Item** that is not available due to previous checkout by a different borrower a request can be made (a reservation) that locks the borrower in as the next person able to check out the **Item**.

Actors

Borrower - Interactive actor who uses the library to search for **Titles**, make reservations, checkout, and return **Items**.

Librarian - Interactive actor responsible for maintenance of the inventory, acting on behalf of the borrowers, and general support of the library (non-automated as well).

Master Librarian - Interactive actor, themselves a **Librarian**, who is also responsible for maintaining the set of librarians for the system. **Registered User** - Any interactive user for whom the system maintains a system account. This includes borrowers, librarians, and master librarians.

Capabilities include basic login, browsing and searching for **Titles**.

4. Supplementary Specification Document

Objective

The purpose of this document is to define the requirements of the Library system. This document lists the requirements that are not readily captured in the use-cases of the use-case model. The supplementary specification and use-case model together capture a complete set of requirements of the system.

Scope

This supplementary specification defines the non-functional requirements of the system such as reliability, performance, supportability, and security as well as functional requirements that are common across a number of use-cases.

Reference

None

Common Functionalities

- Multiple users must be able to perform their work concurrently
- If the reserved item has been purchased or available, the borrower must be notified

Usability

The desktop user interface shall be Windows NT or Windows 2000 compliant

Reliability

The system shall be 24 hours a day, 7 days a week and not more than 10% down time

Performance

- The system shall support up to 2000 simultaneous users against the central database of any given data
- The system must be able to complete 80% of all transactions within 5 minutes

Supportability: None

Security

- The system must prevent borrowers from changing borrowers information, items information, titles information, and librarians information
- Only Librarian can modify borrowers information, items information, and titles information
- Only Master Librarian can modify librarians information

5. Use – Case Model

Actors

Actor is something external to the system and interacts with the system. Actor may be a human being, device or some other software system.

For Library system, actors might be;

- Librarian
- Borrower

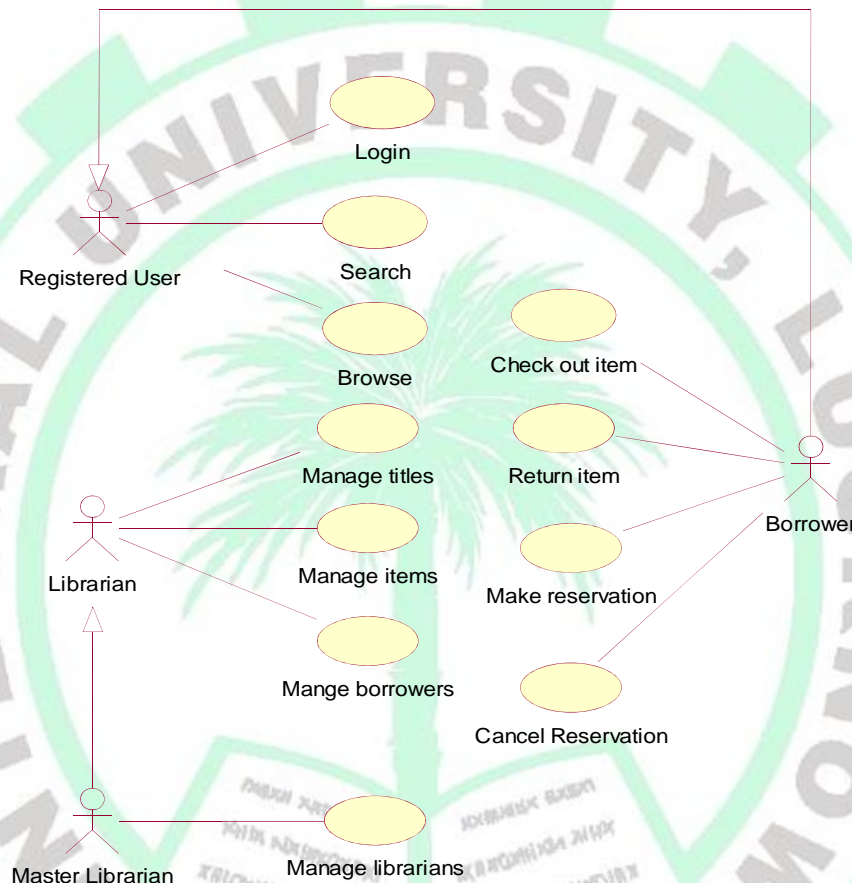
Use – Case

A use-case represents sequence of actions performed by the system that yields an observable result of value for a particular actor. Use-case represents a functional requirement of a system. For Library system, we can find the following use-cases;

- Login
- Search
- Browse
- Check out item
- Return item
- Make reservation
- Cancel reservation

- Manage titles
- Manage items
- Manage borrowers
- Manage librarians

Use - Case Diagram



5.4 Use – Case Descriptions

5.4.1 Use-Case Specification: Login

5.4.1.1 Description

A registered user can log in and, upon verification, can initiate subsequent actions.

5.4.1.2 Flow of Events

5.4.1.2.1 Basic Flow

1. Initiated when a registered user chooses to Login.
2. The system prompts for username and password.
3. The registered user enters a username and password and submits them.
4. The system authenticates the username and password combination.
5. The system authorizes the registered user according to the role(s) to which the registered user has been assigned.
6. The system displays the main page and awaits subsequent action.

5.4.1.2.2 Alternative Flows

- **Invalid Username/Password**
 1. The system displays the Authentication Failed message.
- **Account Locked**

1. The system displays the <appropriate message>.
- **Authentication Service Unavailable**
 1. The system displays a Service Unavailable message and does not permit any further attempts to login.

5.4.1.3 Special Requirements

1. Up to three consecutive failed tries to login with invalid username/password combination until locking an account.
2. Minimum password length is 8 characters, and must include a combination of characters including at least one non-alphabetic character.

5.4.1.4 Preconditions

User has an account with the system

5.4.1.5 Post-conditions

5.4.1.5.1 Primary Success Post-condition

The user is considered authenticated and is viewing the main page from which additional actions can be initiated.

5.4.1.5.2 Login Failure

If the Login fails as described in any of the alternatives above, an appropriate message is displayed and the user is not considered authenticated.

5.4.1.6 Notes

1. So far we are not doing much with roles.
2. The “appropriate message” above is vague; we need to come up with how we report this to the user.
3. We need to talk to security people about how reasonable it is to lock the user account after some number of failed attempts. If we keep that rule, we’ll need an Unlock Account use case.

5.4.2 Use-Case Specification: Browse

5.4.2.1 Description

A registered user can browse the contents of the library as a precursor to other actions.

5.4.3 Flow of Events

5.4.3.1 Basic Flow

1. Initiated when a registered user chooses to browse *Titles*.
2. The system responds by displaying all of the *Titles* in the system, along with topical descriptions.
3. The registered user selects a *Title* for further information.
4. The system displays *Title* detail along with the *Items* and the available action on each *Item*.

5.4.3.2 Alternative Flows

▪ No records

1. The system displays message indicating no *Titles* are in the system.

5.4.4 Special Requirements

1. The *Titles* will be sorted alphabetically by the name.

5.4.5 Preconditions

The user has been authenticated.

5.4.6 Post-conditions

5.4.6.1 Primary Success Post-condition

The registered user is viewing a *Title* along with the related *Items*.

5.4.3 Use-Case Specification: Search

5.4.3.1 Description

A registered user can search the contents of the library as a precursor to other actions.

5.4.3.2 Flow of Events

5.4.3.2.1 Basic Flow

1. Initiated when a registered user chooses to perform a search of *Titles*.
2. The system responds by providing the registered user a means to enter search criteria.
3. The registered user enters search criteria and initiates the query.
4. The system determines results and displays the matching *Titles*, along with topical descriptions.
5. The registered user selects a *Title* for further information.
6. The system displays *Title* detail along with the *Items* and the available action on each *Item*.

5.4.3.2.2 Alternative Flows

▪ No matches

1. The system displays message indicating no *Titles* in the system match this criteria.

5.4.3.3 Special Requirements

1. The search only searches based on the name of the *Item*, not description or any other field.
2. The system shall use the percent sign as a wildcard (in keeping with standard SQL idioms).
3. The results will be sorted alphabetically by the name.

5.4.3.4 Preconditions

The user has been authenticated.

5.4.3.5 Post-conditions

5.4.3.5.1 Primary Success Post-condition

The registered user is viewing a *Title* along with the related *Items*.

5.4.3.6 Notes

1. We might want to combine this with the Search use case. The combined use case could be called Select *Title* and one of the original use cases could be the basic flow and the other would be the alternative.

5.4.4 Use-Case Specification: Make Reservation

5.4.4.1 Description

This use-case starts when the user wants to make a reservation for an item

5.4.4.2 Flow of Events

5.4.4.2.1 Basic flow

1. The system prompts the borrower to enter the item information for which he wants reservation
2. The borrower submits the item information
3. The system marks the item as reserved and associates the borrower with the reservation

5.4.4.2.2 Alternative Flow

None

5.4.4.3 Special requirements

None

5.4.4.4 Pre-conditions

The borrower is viewing a particular title with an item that is not currently available

5.4.4.5 Post-conditions

The item is marked as reserved and the reservation is saved in the database

5.4.4.6 Notes

1. So far there is no nice way to figure out what a borrower has reserved.

5.4.5 Use-Case Specification: Remove Reservation

5.4.5.1 Description

The borrower can remove an existing reservation for an item.

5.4.5.2 Flow of events

5.4.5.2.1 Basic Flow

1. The system prompts the borrower for the item information for which the reservation is removed
2. The borrower enters the item information and submits
3. System marks the item as no longer reserved

5.4.5.2.2 Alternative Flows

None

5.4.5.3 Special requirements

None

5.4.5.4 Pre-conditions

The borrower is viewing a particular *Title* with an *Item* that is reserved by the borrower.

5.4.5.5 Post-conditions

The previously reserved *Item* is no longer reserved.

5.4.6 Use-Case Specification: Check out Item

5.4.6.1 Description

This use-case starts when the borrower wishes to check out an item from the library

5.4.6.2 Flow of Events

5.4.6.2.1 Basic Flow

1. The borrower performs a search for the desired titles
2. The system prompts the borrower to enter search criteria
3. The borrower specifies the search criteria and submits
4. The system locates matching titles and displays them to the borrower
5. The borrower selects titles to check out
6. The system displays the details of titles as well as whether or not there is an available item to be checked out
7. The borrower confirms the check out
8. the system checks out the item
9. Steps 1-8 can be repeated as often as needed by the borrower
10. The borrower completes the check out
11. The system notifies the Librarian that the borrower has concluded the check out item session and displays instructions for the borrower to collect the items

5.4.6.2.2 Alternative Flows

None

5.4.6.3 Special requirements

5.4.6.4 Pre-conditions

The borrower is viewing a particular *Title* with an *Item* that is currently available.

5.4.6.5 Post-conditions

The *Item* is demarked as checked out to the borrower.

5.4.7 Use-Case Specification: Return Item

5.4.7.1 Description

This use-case starts when the borrower wishes to return an item

5.4.7.2 Flow of Events

5.4.7.2.1 Basic Flow

1. The system prompts the borrower to enter the item information he wants to return
2. The borrower enters the item information and submits
3. The system marks the item as available

5.4.7.2.2 Alternative Flows

None

5.4.7.3 Special requirements

None

5.4.7.4 Pre-conditions

The borrower is viewing a particular *Title* with an *Item* that is checked out by the borrower.

5.4.7.5 Post-conditions

The *Item* is demarked as available.

5.4.7.6 Notes

A reasonable future enhancement would be to notify anyone with a reservation on the *Item*.

Exercise:

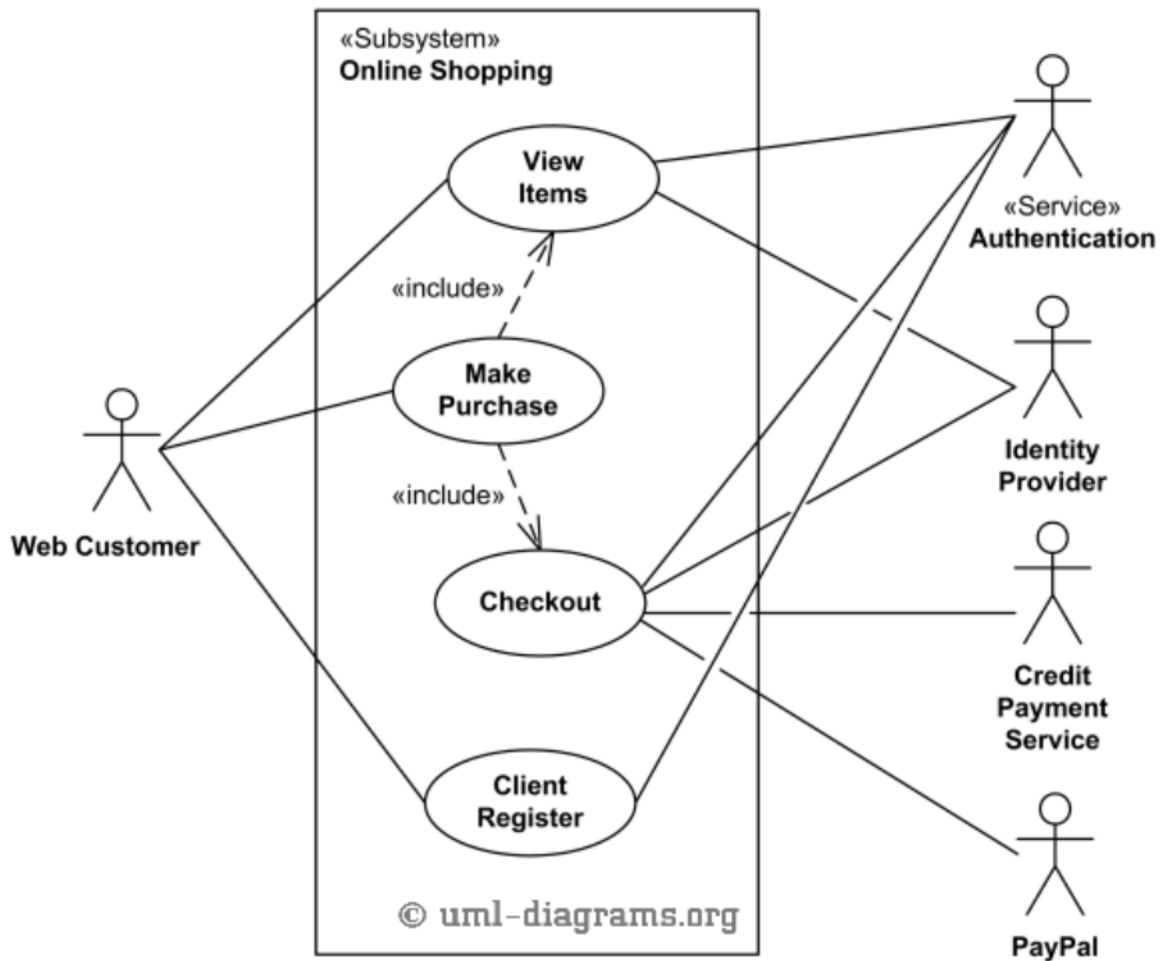
1. Design use case diagram for online shopping. With detail description of use case.
Or

Design use case diagram for ATM system. With detail description of use case.

Solution: Online Shopping

Web Customer actor uses some web site to make purchases online. Top level use cases are **View Items**, **Make Purchase** and **Client Register**. View Items use case could be used by customer as top level use case if customer only wants to find and see some products. This use case could also be used as a part of Make Purchase use case. Client Register use case allows customer to register on the web site, for example to get some coupons or be invited to private sales. Note, that **Checkout** use case is included use case not available by itself - checkout is part of making purchase.

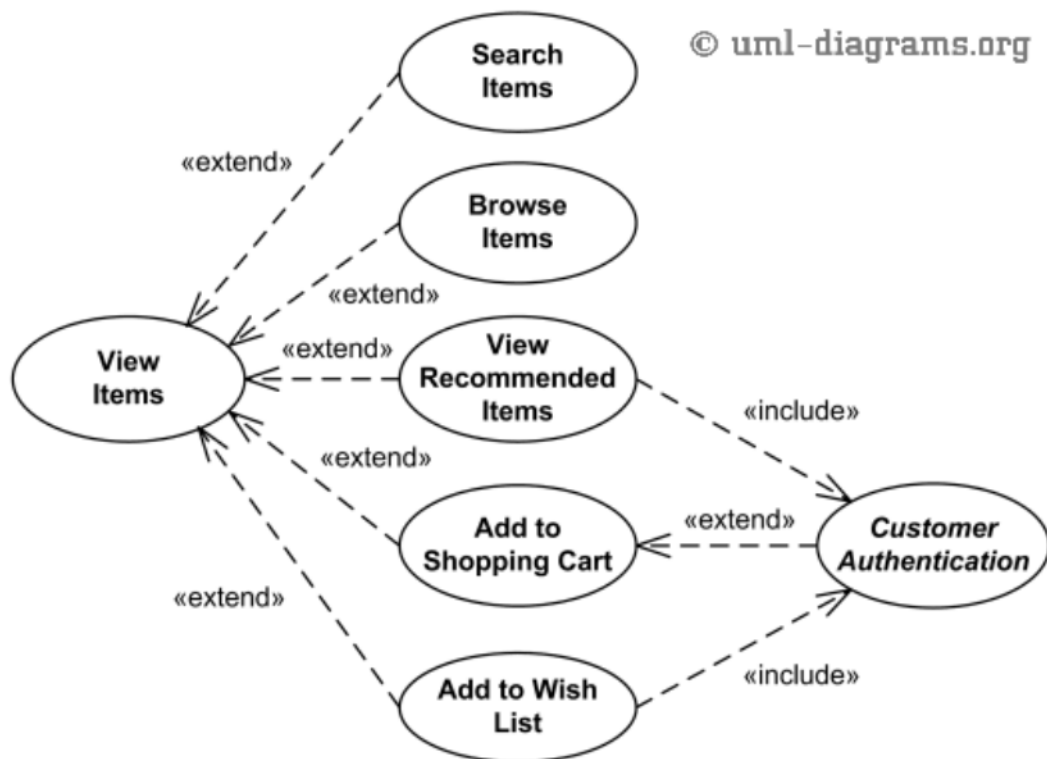
Except for the Web Customer actor there are several other actors which will be described below with detailed use cases.



Online Shopping - Top Level Use Cases

View Items use case is extended by several optional use cases - customer may search for items, browse catalog, view items recommended for him/her, add items to shopping cart or wish list. All these use cases are extending use cases because they provide some optional functions allowing customer to find item.

Customer Authentication use case is included in **View Recommended Items** and **Add to Wish List** because both require customer to be authenticated. At the same time, item could be added to the shopping cart without user authentication.

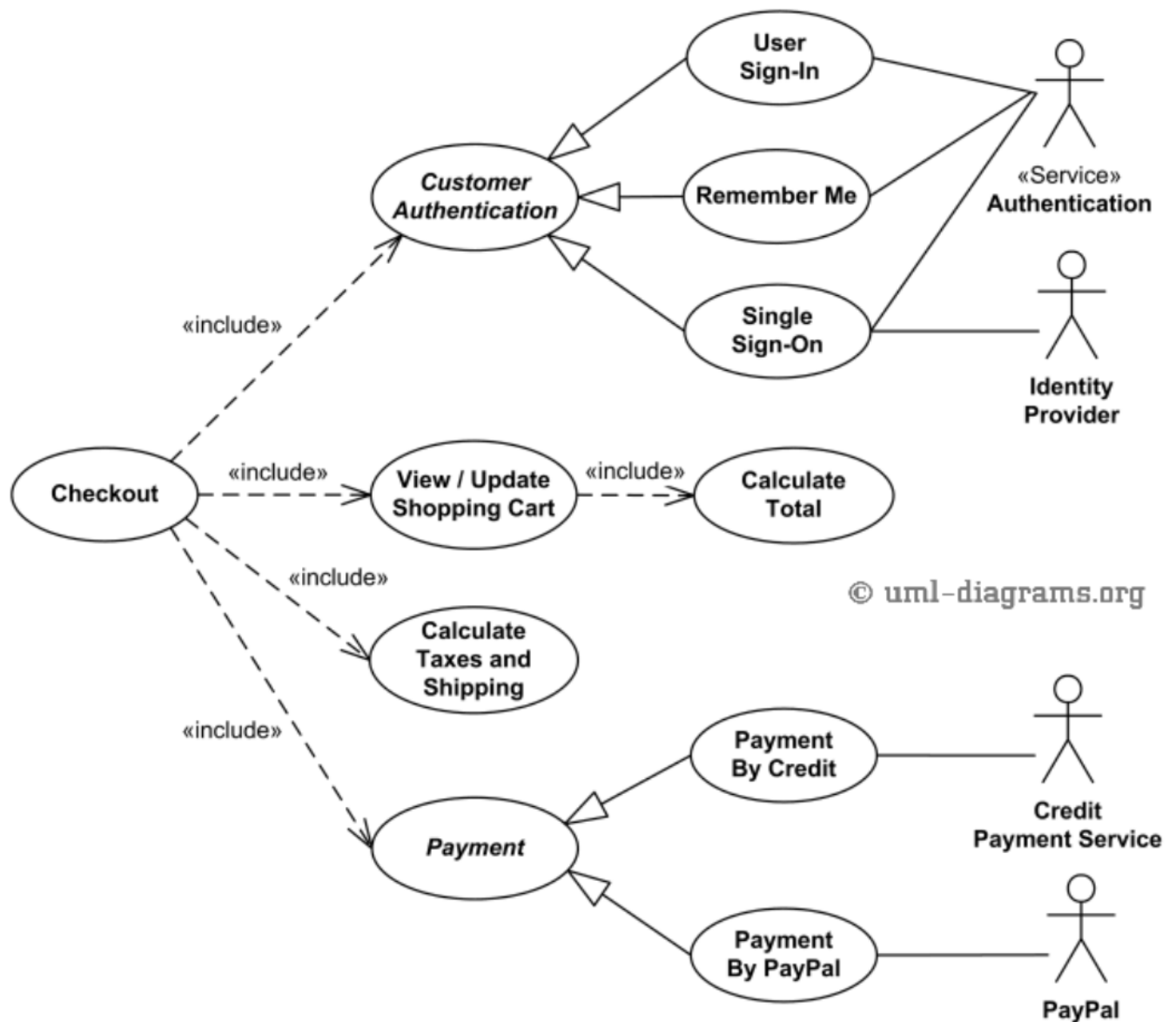


Online Shopping - View Items Use Case

Checkout use case includes several required uses cases. Web customer should be authenticated. It could be done through user login page, user authentication cookie ("Remember me") or Single Sign-On (SSO). Web site authentication service is used in all these use cases, while SSO also requires participation of external identity provider.

Checkout use case also includes Payment use case which could be done either by using credit card and external credit payment service or with PayPal.





Online Shopping - Checkout, Authentication and Payment Use Cases

Bibliography

1. <http://homepages.uel.ac.uk/D.Bowden/index.htm>
2. www.uml.org.cn/umltools/psf/UsingRationalRose.pdf

Experiment No 7

Objective: UML Class Diagram

- Study about
- Design use case diagram for given problem

Time Required

CASE Tool: MS VISIO/Rational Rose

The class diagram is a static diagram. It represents the static view of an application. Class diagram is not only used for visualizing, describing and documenting different aspects of a system but also for constructing executable code of the software application.

The class diagram describes the attributes and operations of a class and also the constraints imposed on the system. The class diagrams are widely used in the modelling of object oriented systems because they are the only UML diagrams which can be mapped directly with object oriented languages.

The class diagram shows a collection of classes, interfaces, associations, collaborations and constraints. It is also known as a structural diagram.

Modeling steps for Class Diagrams

1. Identity the things that are interacting with class diagram.
2. Set the attributes and operations.
3. Set the responsibilities.
4. Identify the generalization and specification classes.
5. Set the relationship among all the things.
6. Adorn with tagged values, constraints and notes.

How to draw Class Diagram?

Class diagrams are the most popular UML diagrams used for construction of software applications. So it is very important to learn the drawing procedure of class diagram.

Class diagrams have lot of properties to consider while drawing but here the diagram will be considered from a top level view.

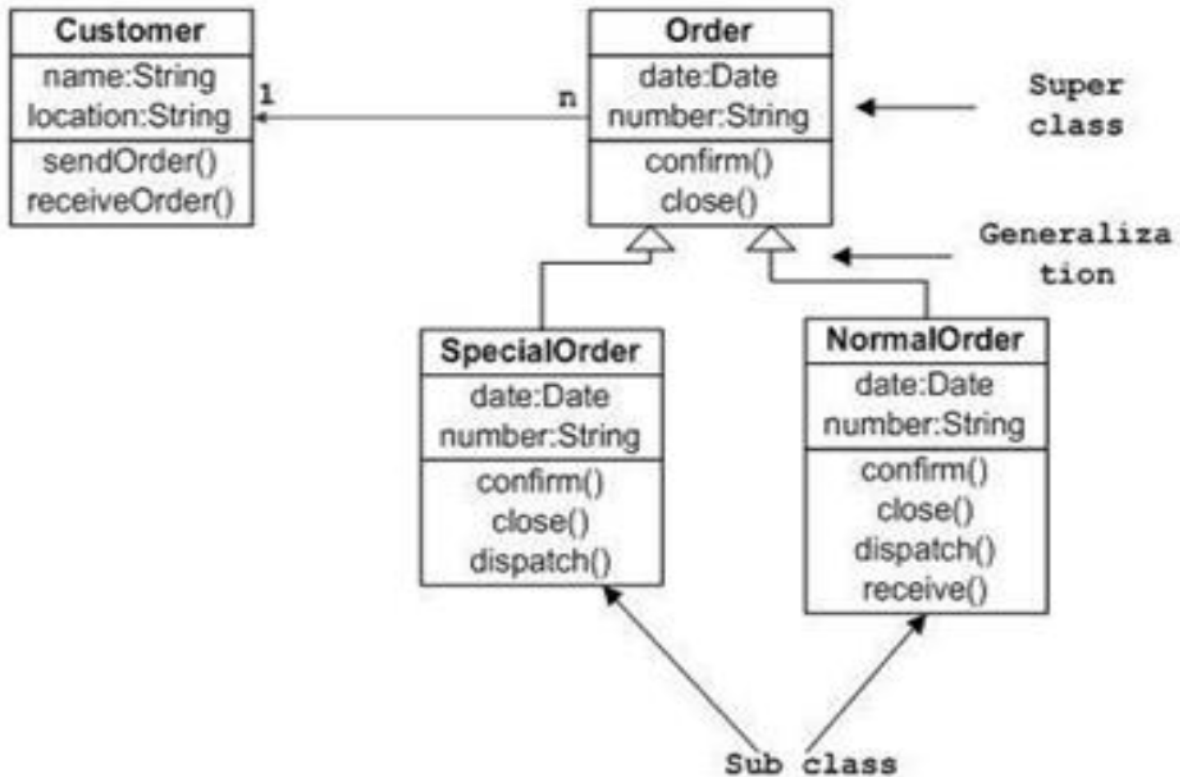
Class diagram is basically a graphical representation of the static view of the system and represents different aspects of the application. So a collection of class diagrams represent the whole system.

The following points should be remembered while drawing a class diagram:

- The name of the class diagram should be meaningful to describe the aspect of the system.
- Each element and their relationships should be identified in advance.
- Responsibility (attributes and methods) of each class should be clearly identified.
- For each class minimum number of properties should be specified. Because unnecessary properties will make the diagram complicated.
- Use notes when ever required to describe some aspect of the diagram. Because at the end of the drawing it should be understandable to the developer/coder.
- Finally, before making the final version, the diagram should be drawn on plain paper and rework as many times as possible to make it correct.

Now the following diagram is an example of an Order System of an application. So it describes a particular aspect of the entire application.

- First of all Order and Customer are identified as the two elements of the system and they have a one to many relationship because a customer can have multiple orders.
- We would keep Order class as an abstract class and it has two concrete classes (inheritance relationship) Special Order and Normal Order.
- The two inherited classes have all the properties as the Order class. In addition they have additional functions like dispatch () and receive ().



So the following class diagram has been drawn considering all the points mentioned above:

Exercise:

1. Design object Diagram for University.
2. Design object diagram for Library.

Bibliography

1. <http://homepages.uel.ac.uk/D.Bowden/index.htm>
2. www.uml.org.cn/umltools/psf/UsingRationalRose.pdf

Experiment No. 8

Objective:

- Study about object Diagram
- Design object diagram for given problem

CASE Tool: MS VISIO/Rational Rose

Object diagrams are derived from class diagrams so object diagrams are dependent upon class diagrams.

Object diagrams represent an instance of a class diagram. The basic concepts are similar for class diagrams and object diagrams. Object diagrams also represent the static view of a system but this static view is a snapshot of the system at a particular moment.

Object diagrams are used to render a set of objects and their relationships as an instance.

Modeling steps for Object Diagrams

Identify the mechanisms which you would like to model.

Identify the classes, use cases, interface, subsystem which are collaborated with mechanisms.

Identify the relationship among all objects.

Walk through the scenario until to reach the certain point and identify the objects at that point.

Render all these classes as objects in diagram.

Specify the links among all these objects.

Set the values of attributes and states of objects.

Purpose:

The purpose of a diagram should be understood clearly to implement it practically. The purposes of object diagrams are similar to class diagrams.

The difference is that a class diagram represents an abstract model consists of classes and their relationships. But an object diagram represents an instance at a particular moment which is concrete in nature.

It means the object diagram is more close to the actual system behaviour. The purpose is to capture the static view of a system at a particular moment. So the purpose of the object diagram can be summarized as:

Forward and reverse engineering.

- Object relationships of a system . Static view of an interaction.
- Understand object behaviour and their relationship from practical perspective.

How to draw Object Diagram?

We have already discussed that an object diagram is an instance of a class diagram. It implies that an object diagram consists of instances of things used in a class diagram.

So both diagrams are made of same basic elements but in different form. In class diagram elements are in abstract form to represent the blue print and in object diagram the elements are in concrete form to represent the real world object.

To capture a particular system, numbers of class diagrams are limited. But if we consider object diagrams then we can have unlimited number of instances which are unique in nature. So only those instances are considered which are having impact on the system.

From the above discussion it is clear that a single object diagram cannot capture all the necessary instances or rather cannot specify all objects of a system. So the solution is:

First, analyze the system and decide which instances are having important data and association.

Second, consider only those instances which will cover the functionality.

Third, make some optimization as the numbers of instances are unlimited.

Before drawing an object diagrams the following things should be remembered and understood clearly:

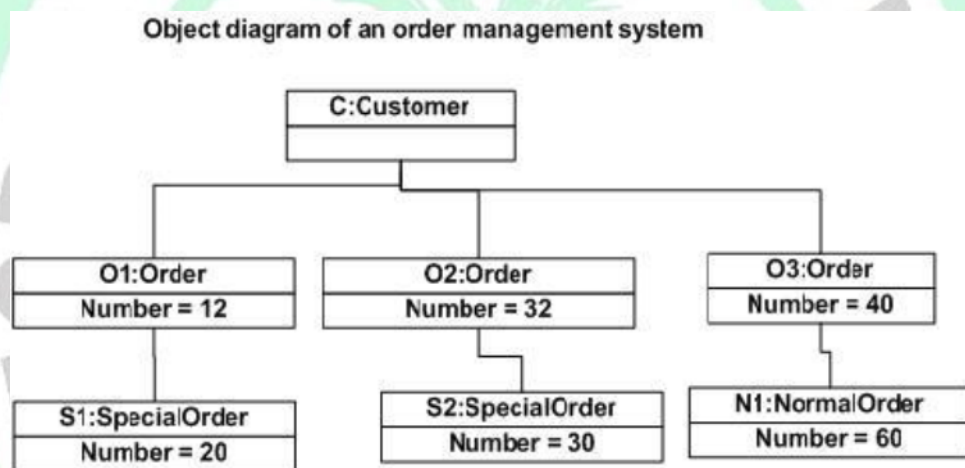
- Object diagrams are consist of objects.
- The link in object diagram is used to connect objects.
- Objects and links are the two elements used to construct an object diagram.

Now after this the following things are to be decided before starting the construction of the diagram: The object diagram should have a meaningful name to indicate its purpose. The most important elements are to be identified.

- The association among objects should be clarified.
- Values of different elements need to be captured to include in the object diagram.
- Add proper notes at points where more clarity is required.

The following diagram is an example of an object diagram. It represents the Order management system which we have discussed in Class Diagram. The following diagram is an instance of the system at a particular time of purchase. It has the following objects

- Customer
- Order
- SpecialOrder



- Normal Order

Now the customer object (C) is associated with three order objects (O1, O2 and O3). These order objects are associated with special order and normal order objects (S1, S2 and N1). The customer is having the following three orders with different numbers (12, 32 and 40) for the particular time considered.

Now the customer can increase number of orders in future and in that scenario the object diagram will reflect that. If order, special order and normal order objects are observed then we you will find that they are having some values.

For orders the values are 12, 32, and 40 which implies that the objects are having these values for the particular moment (here the particular time when the purchase is made is considered as the moment) when the instance is captured.

The same is for special order and normal order objects which are having number of orders as 20, 30 and 60. If a different time of purchase is considered then these values will change accordingly. So the following object diagram has been drawn considering all the points mentioned above:

Exercise:

1. Design object Diagram for University.
2. Design object diagram for Library.

Bibliography

1. <http://homepages.uel.ac.uk/D.Bowden/index.htm>
2. www.uml.org.cn/umltools/psf/UsingRationalRose.pdf

Experiment - 9

Objective:

- Study about Deployment Diagram
- Design Deployment diagram for given problem

CASE Tool: MS VISIO/Rational Rose

UML Deployment Diagram

Deployment diagrams are used to visualize the topology of the physical components of a system where the software components are deployed. So deployment diagrams are used to describe the static deployment view of a system. Deployment diagrams consist of nodes and their relationships.

Purpose:

The name Deployment itself describes the purpose of the diagram. Deployment diagrams are used for describing the hardware components where software components are deployed. Component diagrams and deployment diagrams are closely related. Component diagrams are used to describe the components and deployment diagrams shows how they are deployed in hardware.

UML is mainly designed to focus on software artifacts of a system. But these two diagrams are special diagrams used to focus on software components and hardware components.

So most of the UML diagrams are used to handle logical components but deployment diagrams are made to focus on hardware topology of a system. Deployment diagrams are used by the system engineers.

The purpose of deployment diagrams

Visualize hardware topology of a system.

Describe the hardware components used to deploy software components.

Describe runtime processing nodes.

Modeling steps for Deployment Diagram

Identify the processors which represent client & server.

Provide the visual cue via stereotype classes.

Group all the similar clients into one package.

Provide the links among clients & servers.

Provide the attributes & operations.

Specify the components which are living on nodes.

Adorn with nodes & constraints & draw the deployment diagram.

How to draw Deployment Diagram? Deployment diagram represents the deployment view of a system. It is related to the component diagram. Because the components are deployed using the deployment diagrams. A deployment diagram consists of nodes. Nodes are nothing but physical hardware used to deploy the application.

Deployment diagrams are useful for system engineers. An efficient deployment diagram is very important because it controls the following parameters

- Performance

- Scalability
- Maintainability
- Portability

So before drawing a deployment diagram the following artifacts should be identified:

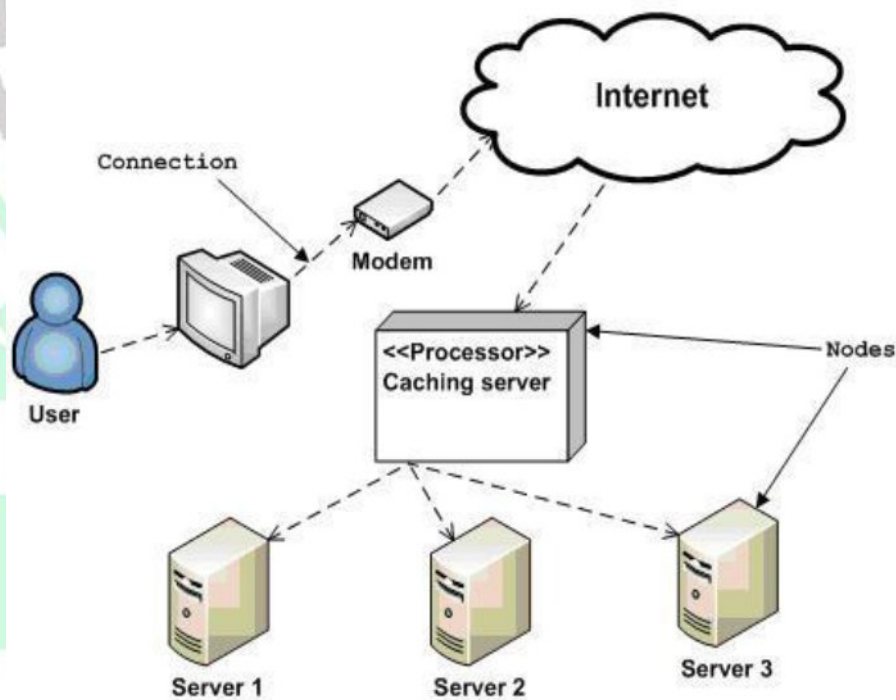
- Nodes
- Relationships among nodes

The following deployment diagram is a sample to give an idea of the deployment view of order management system. Here we have shown nodes as:

- Monitor
- Modem
- Caching server
- Server

The application is assumed to be a web based application which is deployed in a clustered environment using server 1, server 2 and server 3. The user is connecting to the application using internet. The control is flowing from the caching server to the clustered environment. So the following deployment diagram has been drawn considering all the points mentioned above:

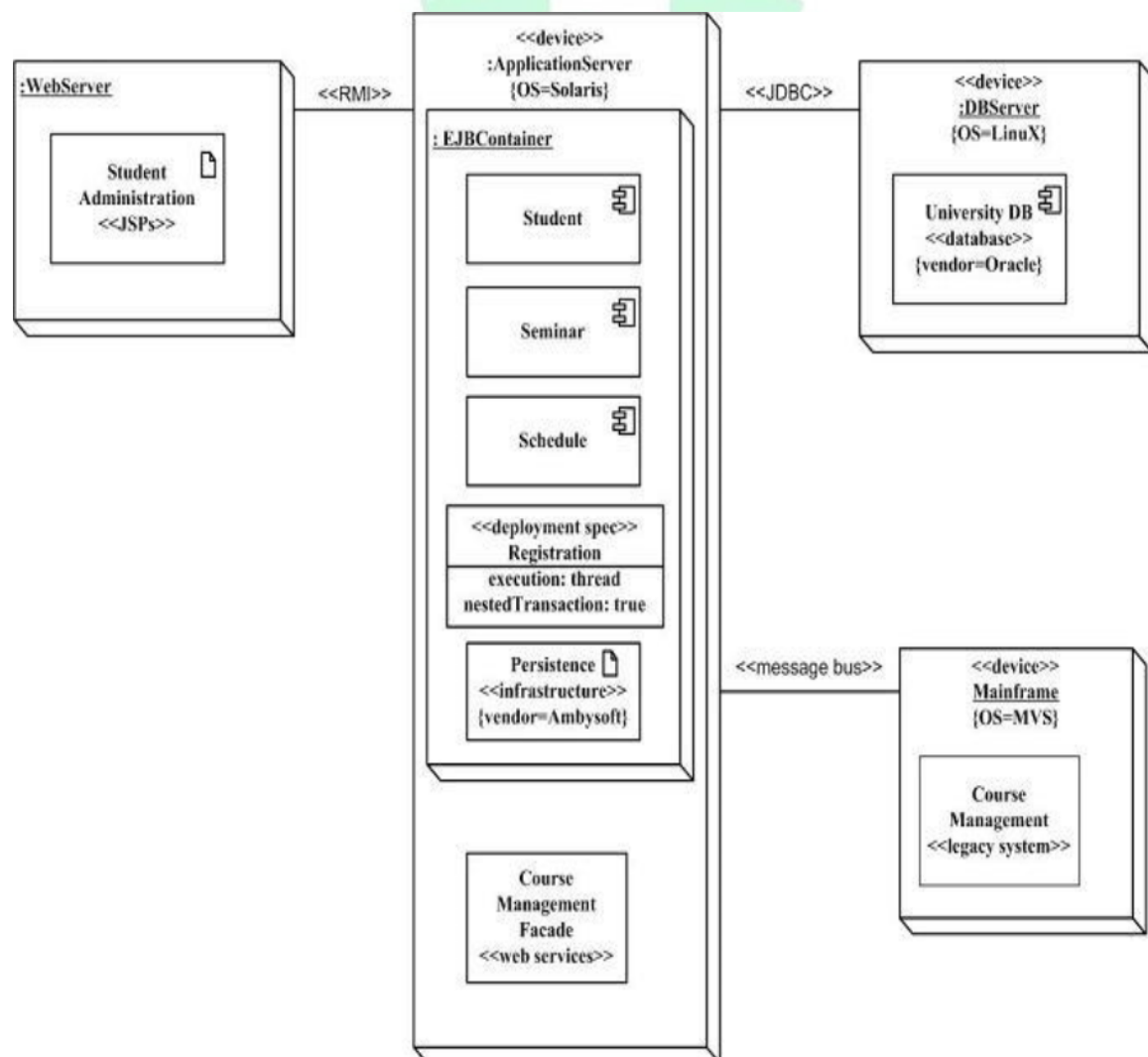
Deployment diagram of an order management system



Exercise:

1. Design deployment diagram for the university information system.

Solution:



Bibliography

1. <http://homepages.uel.ac.uk/D.Bowden/index.htm>
2. www.uml.org.cn/umltools/psf/UsingRationalRose.pdf

Experiment - 10

Objective:

Activity Diagrams

Study about Activity Diagrams

Design Activity Diagrams for given problem

CASE Tool: MS VISIO/Rational Rose

Overview:

Activity diagram is another important diagram in UML to describe dynamic aspects of the system.

Activity diagram is basically a flow chart to represent the flow from one activity to another activity. The activity can be described as an operation of the system.

So the control flow is drawn from one operation to another. This flow can be sequential, branched or concurrent. Activity diagrams deal with all types of flow control by using different elements like fork, join etc.

Purpose:

The basic purposes of activity diagrams are similar to other four diagrams. It captures the dynamic behaviour of the system. Other four diagrams are used to show the message flow from one object to another but activity diagram is used to show message flow from one activity to another.

Activity is a particular operation of the system. Activity diagrams are not only used for visualizing dynamic nature of a system but they are also used to construct the executable system by using forward and reverse engineering techniques. The only missing thing in activity diagram is the message part.

It does not show any message flow from one activity to another. Activity diagram is sometimes considered as the flow chart. Although the diagram looks like a flow chart but it is not. It shows different flows like parallel, branched, concurrent and single.

So the purposes can be described as:

Draw the activity flow of a system.

Describe the sequence from one activity to another.

Describe the parallel, branched and concurrent flow of the system.

How to draw Activity Diagram?

Activity diagrams are mainly used as a flow chart consists of activities performed by the system. But activity diagrams are not exactly a flow chart as they have some additional capabilities. These additional capabilities include branching, parallel flow, swimlane etc.

Before drawing an activity diagram we must have a clear understanding about the elements used in activity diagrams. The main element of an activity diagram is the activity itself. An activity is a function performed by the system. After identifying the activities we need to understand how they are associated with constraints and conditions.

So before drawing an activity diagram we should identify the following elements:

- Activities
- Association
- Conditions
- Constraints

Once the above mentioned parameters are identified we need to make a mental layout of the entire flow. This mental layout is then transformed into an activity diagram.

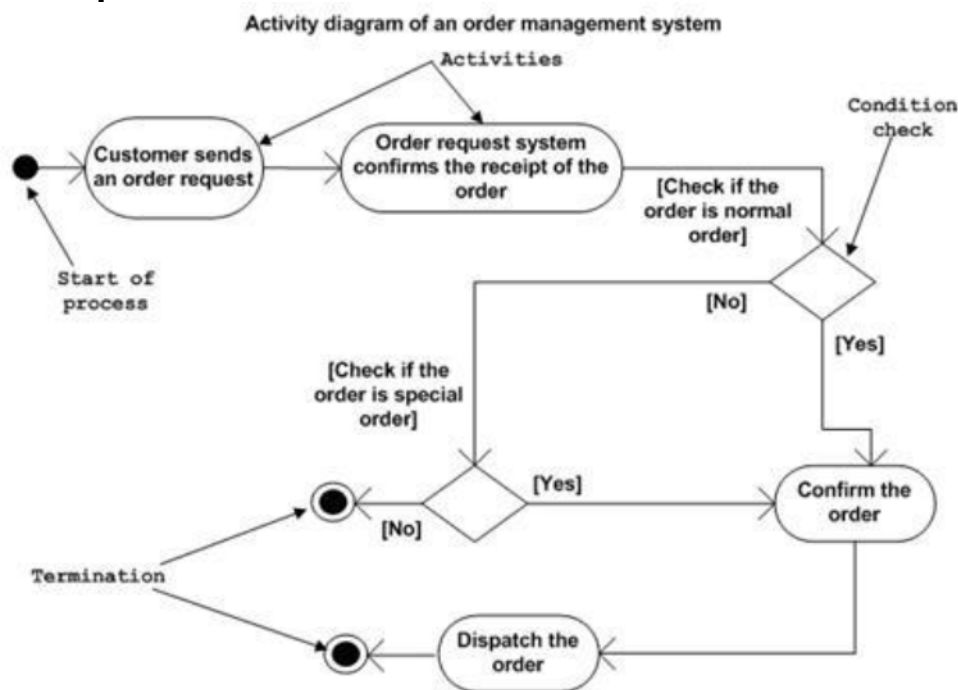
The following is an example of an activity diagram for order management system. In the diagram four activities are identified which are associated with conditions. One important point should be clearly understood that an activity diagram cannot be exactly matched with the code. The activity diagram is made to understand the flow of activities and mainly used by the business users.

The following diagram is drawn with the four main activities:

- Send order by the customer
- Receipt of the order
- Confirm order
- Dispatch order

After receiving the order request condition checks are performed to check if it is normal or special order. After the type of order is identified dispatch activity is performed and that is marked as the termination of the process.

Example



Exercise:

1. Design Activity diagram for the ATM system
2. Design Activity diagram for the Online Shopping system.

Bibliography

1. <http://homepages.uel.ac.uk/D.Bowden/index.htm>
2. www.uml.org.cn/umltools/psf/UsingRationalRose.pdf

Experiment - 11

Objective:

UML State chart Diagram

Study about State chart Diagram

Design State chart Diagram for given problem

CASE Tool: MS VISIO/Rational Rose

Overview:

The name of the diagram itself clarifies the purpose of the diagram and other details. It describes different states of a component in a system. The states are specific to a component/object of a system.

A Statechart diagram describes a state machine. Now to clarify it state machine can be defined as a machine which defines different states of an object and these states are controlled by external or internal events.

Activity diagram explained in next chapter, is a special kind of a Statechart diagram. As Statechart diagram defines states it is used to model lifetime of an object.

Purpose:

Statechart diagram is one of the five UML diagrams used to model dynamic nature of a system. They define different states of an object during its lifetime. And these states are changed by events. So Statechart diagrams are useful to model reactive systems. Reactive systems can be defined as a system that responds to external or internal events.

Statechart diagram describes the flow of control from one state to another state. States are defined as a condition in which an object exists and it changes when some event is triggered. So the most important purpose of Statechart diagram is to model life time of an object from creation to termination.

Statechart diagrams are also used for forward and reverse engineering of a system. But the main purpose is to model reactive system.

Following are the main purposes of using Statechart diagrams:

- To model dynamic aspect of a system.
- To model life time of a reactive system.
- To describe different states of an object during its life time.
- Define a state machine to model states of an object.

How to draw Statechart Diagram?

Statechart diagram is used to describe the states of different objects in its life cycle. So the emphasis is given on the state changes upon some internal or external events. These states of objects are important to analyze and implement them accurately.

Statechart diagrams are very important for describing the states. States can be identified as the condition of objects when a particular event occurs.

Before drawing a Statechart diagram we must have clarified the following points:

- Identify important objects to be analyzed.
- Identify the states.
- Identify the events.

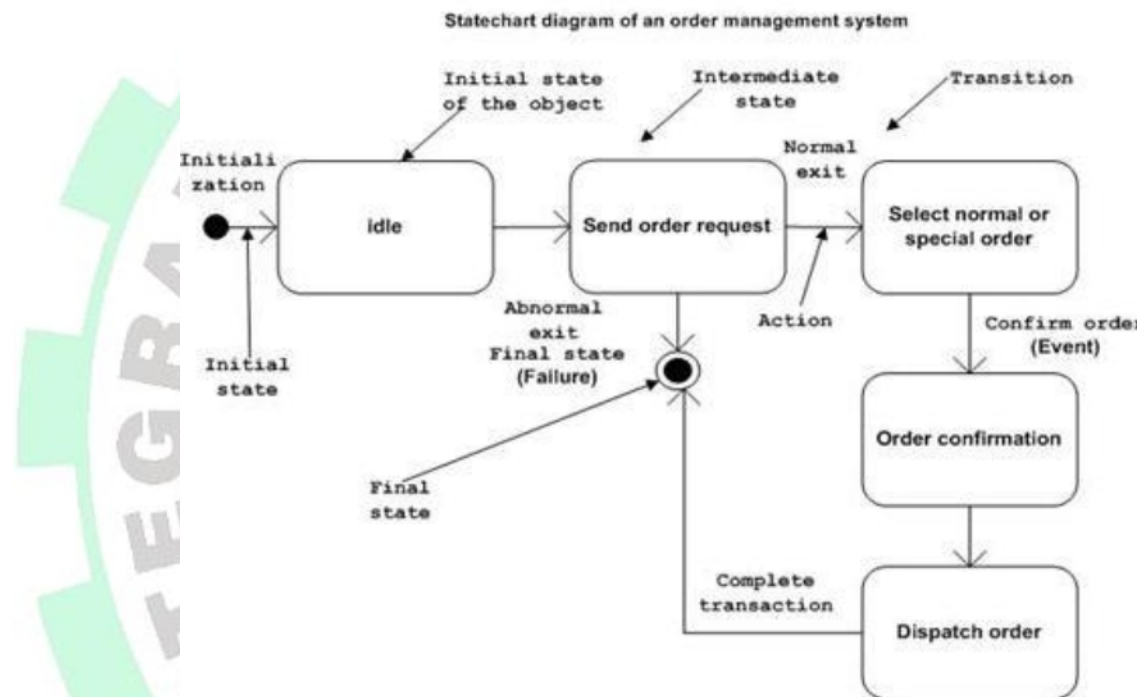
The following is an example of a Statechart diagram where the state of Order object is analyzed.

The first state is an idle state from where the process starts. The next states are arrived for events like send request, confirm request, and dispatch order. These events are responsible for state changes of order object.

During the life cycle of an object (here order object) it goes through the following states and there may be some abnormal exists also. This abnormal exit may occur due to some problem in the system. When the entire life cycle is complete it is considered as the complete transaction as mentioned below.

The initial and final state of an object is also shown below.

Example:



Exercise:

1. Design Statechart diagram for the ATM system
2. Design Statechart diagram for the Online Shopping system.

Bibliography

1. <http://homepages.uel.ac.uk/D.Bowden/index.htm>
2. www.uml.org.cn/umltools/psf/UsingRationalRose.pdf
3. <http://www.tutorialspoint.com/uml/>

Experiment No. 12

Objectives:

Estimation of Project Metrics

Categorize projects using COCOMO, and estimate effort and development time required for a project

Estimate the program complexity and effort required to recreate it using Halstead's metrics

Project Estimation Techniques

A software project is not just about writing a few hundred lines of source code to achieve a particular objective. The scope of a software project is comparatively quite large, and such a project could take several years to complete. However, the phrase "quite large" could only give some (possibly vague) qualitative information. As in any other science and engineering discipline, one would be interested to measure how complex a project is. One of the major activities of the project planning phase, therefore, is to estimate various project parameters in order to take proper decisions. Some important project parameters that are estimated include:

Project size: What would be the size of the code written say, in number of lines, files, modules?

Cost: How much would it cost to develop a software? A software may be just pieces of code, but one has to pay to the managers, developers, and other project personnel.

Duration: How long would it be before the software is delivered to the clients?

Effort: How much effort from the team members would be required to create the software?

In this experiment we will focus on two methods for estimating project metrics: COCOMO and Halstead's method.

COCOMO

COCOMO (Constructive Cost Model) was proposed by Boehm. According to him, there could be three categories of software projects: organic, semidetached, and embedded. The classification is done considering the characteristics of the software, the development team and environment. These product classes typically correspond to application, utility and system programs, respectively. Data processing programs could be considered as application programs. Compilers, linkers, are examples of utility programs. Operating systems, real-time system programs are examples of system programs. One could easily apprehend that it would take much more time and effort to develop an OS than an attendance management system.

The concept of organic, semidetached, and embedded systems are described below.

Organic: A development project is said to be of organic type, if

The project deals with developing a well understood application

The development team is small

The team members have prior experience in working with similar types of projects

Semidetached: A development project can be categorized as semidetached type, if

The team consists of some experienced as well as inexperienced staff

Team members may have some experience on the type of system to be developed

Embedded: Embedded type of development project are those, which

Aims to develop a software strongly related to machine hardware

Team size is usually large

Boehm suggested that estimation of project parameters should be done through three stages:

Basic COCOMO, Intermediate COCOMO, and Complete COCOMO.

Basic COCOMO Model

The basic COCOMO model helps to obtain a rough estimate of the project parameters. It estimates effort and time required for development in the following way:

Effort = a * (KDSI)^b PMT_{dev} = 2.5 * (Effort)^c Months where

KDSI is the estimated size of the software expressed in Kilo Delivered Source Instructions
 a, b, c are constants determined by the category of software project
 Effort denotes the total effort required for the software development, expressed in person months (PMs)
 T_{dev} denotes the estimated time required to develop the software (expressed in months)
 The value of the constants a, b, c are given below:

Software project	a	b	c
Organic	2.4	1.05	0.38
Semi-detached	3.0	1.12	0.35
Embedded	3.6	1.20	0.32

Intermediate COCOMO Model

The basic COCOMO model considers that effort and development time depends only on the size of the software. However, in real life there are many other project parameters that influence the development process. The intermediate COCOMO take those other factors into consideration by defining a set of 15 cost drivers (multipliers) as shown in the table below [i]. Thus, any project that makes use of modern programming practices would have lower estimates in terms of effort and cost. Each of the 15 such attributes can be rated on a six-point scale ranging from "very low" to "extra high" in their relative order of importance. Each attribute has an effort multiplier fixed as per the rating. The product of effort multipliers of all the 15 attributes gives the **Effort Adjustment Factor (EAF)**.

Cost drivers for INtermediate COCOMO (Source: http://en.wikipedia.org/wiki/COCOMO)						
Cost Drivers	Ratings					
	Very Low	Low	Nominal	High	Very High	Extra High
Product attributes						
Required software reliability	0.75	0.88	1.00	1.15	1.40	
Application database		0.94	1.00	1.08	1.16	
Complexity of the product	0.70	0.85	1.00	1.15	1.30	1.65
Hardware attributes						

Cost drivers for Intermediate COCOMO
(Source: <http://en.wikipedia.org/wiki/COCOMO>)

Cost Drivers	Ratings					
	Very Low	Low	Nominal	High	Very High	Extra High
Run-time performance constraints			1.00	1.11	1.30	1.66
Memory constraints			1.00	1.06	1.21	1.56
Volatility of the virtual machine environment		0.87	1.00	1.15	1.30	
Required turnabout time		0.87	1.00	1.07	1.15	
Personnel attributes						
Analyst capability	1.46	1.19	1.00	0.86	0.71	
Applications experience	1.29	1.13	1.00	0.91	0.82	
Software engineer capability	1.42	1.17	1.00	0.86	0.70	
Virtual machine experience	1.21	1.10	1.00	0.90		
Programming language experience	1.14	1.07	1.00	0.95		
Project attributes						
Application of software engineering methods	1.24	1.10	1.00	0.91	0.82	
Use of software tools	1.24	1.10	1.00	0.91	0.83	
Required development schedule	1.23	1.08	1.00	1.04	1.10	

EAF is used to refine the estimates obtained by basic COCOMO as follows: $\text{Effort}_{\text{corrected}} = \text{Effort} * \text{EAF}$
 $\text{dev}_{\text{corrected}} = 2.5 * (\text{Effort}_{\text{corrected}})^c$

Complete COCOMO Model

Both the basic and intermediate COCOMO models consider a software to be a single homogeneous entity -- an assumption, which is rarely true. In fact, many real life applications are made up of several smaller sub-systems. (One might not even develop all the sub-systems -- just use the available services). The complete COCOMO model takes these factors into account to provide a far more accurate estimate of project metrics.

To illustrate this, consider a very popular distributed application: the ticket booking system of the Indian Railways. There are computerized ticket counters in most of the railway stations of our country. Tickets can be booked / cancelled from any such counter. Reservations for future tickets, cancellation of reserved tickets could also be performed. On a high level, the ticket booking system has three main components:

Database

Graphical User Interface (GUI)

Networking facilities

Among these, development of the GUI is considered as an organic project type; the database module could be considered as a semi-detached software. The networking module can be considered as an embedded software. To obtain a realistic cost, one should estimate the costs for each component separately, and then add it up.

Advantages of COCOMO

COCOMO is a simple model, and should help one to understand the concept of project metrics estimation.

Drawbacks of COCOMO

COCOMO uses KDSI, which is not a proper measure of a program's size. Indeed, estimating the size of a software is a difficult task, and any slight miscalculation could cause a large deviation in subsequent project estimates. Moreover, COCOMO was proposed in 1981 keeping the waterfall model of project life cycle in mind [2]. It fails to address other popular approaches like prototype, incremental, spiral, agile models. Moreover, in present day a software project may not necessarily consist of coding of every bit of functionality. Rather, existing software components are often used and glued together towards the development of a new software. COCOMO is not suitable in such cases.

COCOMO II was proposed later in 2000 to many of address these issues.

Halstead's Complexity Metrics

Halstead took a linguistic approach to determine the complexity of a program. According to him, a computer program consists of a collection of different operands and operators. The definition of operands and operators could, however, vary from one person to another and one programming language to other. Operands are usually the implementation variables or constants -- something upon which an operation could be performed. Operators are those symbols that affects the value of operands. Halstead's metrics are computed based on the operators and operands used in a computer program. Any given program has the following four parameters:

n1: Number of unique operators used in the program

n2: Number of unique operands used in the program

N1: Total number of operators used in the program

N2: Total number of operands used in the program

Using the above parameters one compute the following metrics:

Program Length: $N = N1 + N2$

Program Vocabulary: $n = n1 + n2$

Volume: $V = N * \lg n$

Difficulty: $D = (n1 * N2) / (2 * n2)$

Effort: $E = D * V$

Time to Implement: $T = E / 18$ (in seconds) [vi]

The program volume V is the minimum number of bits needed to encode the program. It represents the size of the program while taking into account the programming language. The difficulty metric indicates how difficult a program is to write or understand. Effort denotes the "mental effort" required to develop the software, or to recreate the same in another programming language [iv].

Exercise:

Considering your immense expertise in software development, The Absolute Beginners Inc. has recently allotted you a mega project. The goal of the project is to create a database of all Hindi films released since 2000. The software would allow one to generate a list of top ten hit films, top ten flop films, best comedy films, and so on. Using your prior experience you have decided the approximate sizes of each module of the software as follows:

Data entry (0.9 KDSI)

Data update (0.7 KDSI)

Query (0.9 KDSI)

Report generation and display (2 KDSI)

Also take into consideration the following cost drivers with their ratings:

Storage constraints (Low)

Experience in developing similar software (High)

Programming capabilities of the developers (High)

Application of software engineering methods (High)

Use of software tools (High)

(All other cost drivers have nominal rating).

Now answer the following:

Applying intermediate COCOMO estimate the minimum size of the team you would require to develop this system

Assuming that your client would pay Rs. 50,000 per month of development, how much would be the likely billing?

Learning Objectives:

Identify type of a project as per COCOMO

Prepare an estimate of required effort and cost

Limitations: Values presented here are arbitrary and doesn't relate to real life

Note: The above example has been adapted from COCOMO (Constructive Cost Model), Seminar on Software Cost Estimation WS 2002 / 2003, presented by Nancy Merlo – Schett.

Bibliography:

Fundamentals of Software Engineering, Rajib Mall, Prentice-Hall of India, 3rd Edition, 2009
Software Engineering, Ian Sommerville, Addison Wesley Longman, 9th Edition, March 2010

Experiment - 13

Objectives:

Automated Functional Testing Using IBM Rational Robot

1. INTRODUCTION

Software testing is an important stage in software life cycle, and it is an assurance of software quality. Software testing exists in each stage of software life cycle, and verifies the expected results are achieved or not, correct the bugs as soon as possible. In software development processing, bugs are always existed no matter what technology is adopted. Testing is applied to find bugs, and used to calculate software bugs density.

Furthermore, the software testing is defined in as "the dynamic verification of the behavior of a program on a finite set of test cases, suitably selected from the usually infinite executions domain, against the expected behavior". However, the software testing process can be assisted with software tools to make it automated.

Types of Software Testing

Although there are many types of software testing, this paper will only include the following types:

Stress Testing: Testing conducted to evaluate a system or component at or beyond the limits of its specified requirements.

Load Testing: A test type concerned with measuring the behavior of a component or system with increasing load.

Regressions Testing: It is the testing which is to be done to software that was previously working correctly and stops working as intended due to changes.

Functional Testing: It is the testing which is conducted on a complete and integrated system to evaluate its compliance with its specified requirements.

Unit Testing: It is the verification and validation technique where the programmer gains confidence that individual units of source code are fit for use.

Performance Testing: It is the testing which refers to the assessment of the performance of a human examinee.

Acceptance Testing: It is the testing which involves running a suite of tests on the completed system. **Security Testing:** It is the testing which determines that an Information System protects data and maintains functionality as intended.

Open Source Testing: It is a functional and unit testing framework for open source software products.

Manual testing is time-consuming, error-prone and requires lot of infrastructure and manpower. All these drawbacks can be overcome if the testing process is automated. The testing tools reduce manual testing to a large extent and the testing can be done automatically.

Why Automating Testing?

Every software development group tests its products, yet delivered software always has defects. Test engineers strive to catch them before the product is released but they always creep in and they often reappear, even with the best manual testing processes. Automated software testing is the best way to increase the effectiveness, efficiency and coverage of software testing process.

What is Rational Robot?

Rational Robot is a complete set of components for automating the testing of Microsoft

Windows client/server and Internet applications. The main component of Robot starts recording tests in as few as two mouse clicks. After recording, Robot plays back the tests in a fraction of the time it would take to repeat the actions manually. Rational Robot is an automated functional regression testing tool.

Components of Rational Robot:

Rational Administrator: Create and manage rational projects to store testing information.

Rational Test Manager: Review and analyze test results.

Object Properties, Text, Grid, and Image Comparators: View and analyze the results of verification point playback.

Rational Site Check: Manage Internet and intranet Web sites.

Case Study

Classics Online is the case study of this research. In this paper we use a simple Visual Basic demo application called Classics Online, included with Rational Robot as Application under Test (AUT) which is shown in figure 1.



Figure 1. Main Screen of Classics Online

Classics Online is a simple Visual Basic application that simulates a classical music shopping application. The main screen consists of a tree control where the buyer can select the CD to purchase.

Rational administrator

This component is used to create and manage projects. The project created in the Rational Administrator is shown in the following figure 2.



Figure 2. Rational Administrator Window

Rational projects store application testing information such as scripts, verification points, queries, and defects. Projects help to organize the testing information and resources for easy tracking.

Starting Rational Robot

Next step is to log onto a Rational Administrator Project. The project just created should be selected. The main screen of Rational Robot opens which is as shown in figure 3.

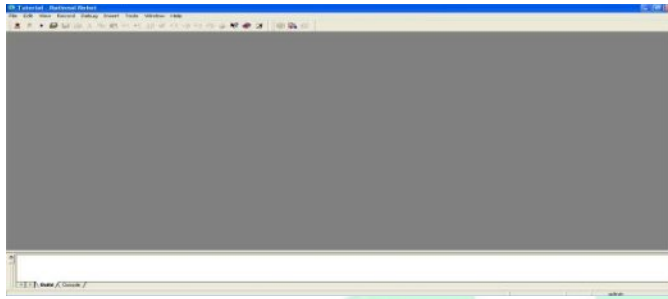


Figure 3. Main screen of Rational Robot

Recording a Test Script

To start recording a GUI test script, click on the red button on the top-left called “Record GUI script” as shown in figure 4.



Figure 4. Record GUI Script button on toolbar

On the screen that pops up, enter a name of the script (OrderItem) and click OK as shown in figure 5.

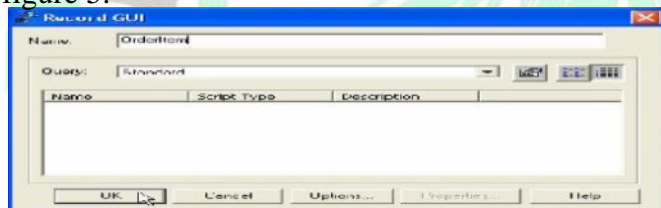


Figure 5. Record GUI dialog box

Rational Robot records everything that can be done on computer -- all clicks, drags, and typing.

Application under Test

The first step in recording the script is to start the Application under Test (AUT). Click the last button on the toolbar – Display GUI Insert Toolbar as shown in figure 6.



Figure 6. Display GUI Insert Toolbar

This toolbar has many options that we use while recording, but for now select the “Start Application” button as shown in figure 7.



Figure 7. Start button on GUI Insert toolbar

Under Application Name enter “C:\ProgramFiles\Rational\RationalTest\SampleApplications\Classics Online\ClassicsA.exe” or browse for it and hit OK as shown in figure 8.

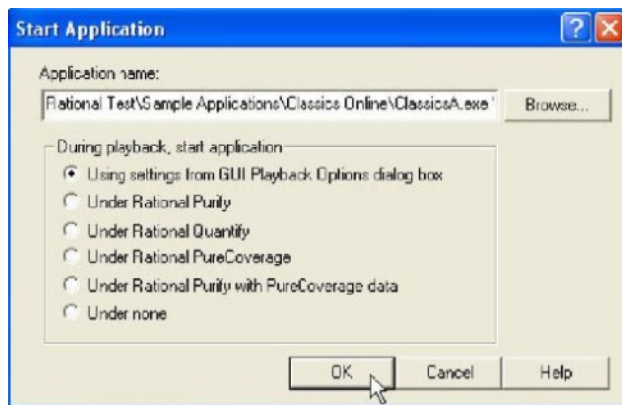


Figure 8. Start Application dialog box

The login window for Classic Online Appears. Click OK as shown in figure 9.

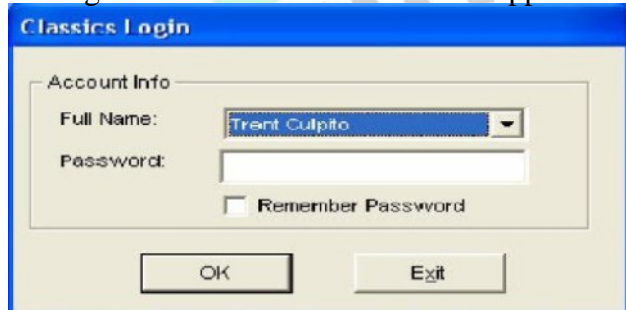


Figure 9. The login window for Classics Online

Once again, select “Beethoven Symphony No.9” and click press here to order as shown in figure 10.



Figure 10. Classics Online window

Verification Points

So far we’ve only had Robot record the navigation through the application, but have not actually tested anything. To test something, we set “Verification points” (VPs) on items we want to check against. Now we must designate the object to be tested. We can do this either by dragging the “Object Finder Tool” over the object, or by browsing for it. Drag and drop the tool over the Customer Name text field as shown in figure 11.

Make An Order

Item: Beethoven - Symphony No. 9	Sub-Total: \$ 15.99
Quantity: <input type="text" value="1"/>	S+H: \$ 2.00
	Total: \$ 17.99

Payment Information

Card Number (include the spaces):

Card Type: Expiration Date:

Your Information

Name:

Street:

City, State Zip:

Telephone:

Figure 11. Make an Order window

Notice that the tool-top provides info about the object as Robot sees it.

Remember we are inserting an object properties verification point. The window that appears next shows us all the properties that we could verify with this verification point as shown in figure 12.

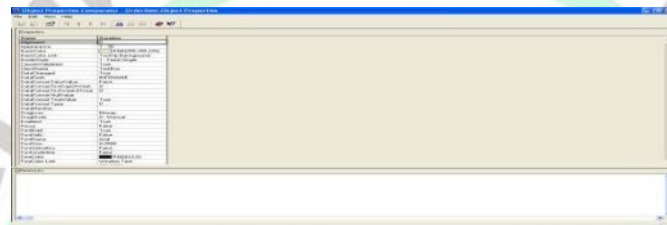


Figure 12. Object Properties Verification Point window

Notice how Rational Robot can see many properties of the object.

The Test Script

Now that we have finished recording, Rational Robot restores itself. Maximize Rational Robot and maximize the script window within Rational Robot. The largest pane is the script itself which is shown in figure 13.

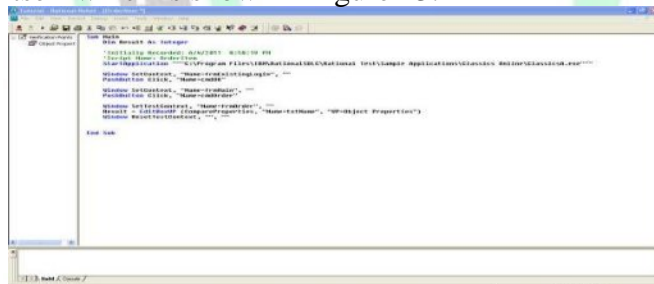


Figure 13. Script Window

The commands we see are in a language called SQABasic. SQABasic is an extension of the Visual Basic programming language. As we can see, it is easy to read and understand what the script will do. Commands like Start Application and Pushbutton Click are pretty self-explanatory.

Running the Test Script

Click on playback script button on the toolbar as shown in figure 14.

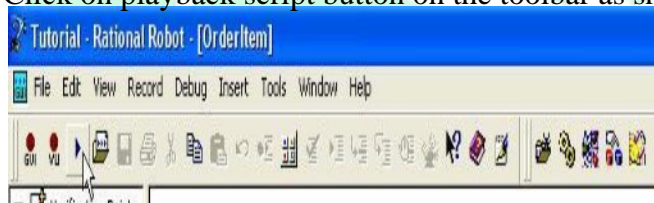


Figure 14. Playback Script button on the toolbar

Rational Robot plays the script much faster than we could manually. Besides the repeatability of an automated testing approach, the computer can perform tests much faster than we ever could manually.

Exercise:-

Text The Sample Code Given in Rational Robot

Bibliography:

1. Paul C.Jorgensen. Software Testing: A Craftsman CRC Press, 2002
2. Duan Nian, Software performance testing and practical. Tsinghua university Press, 2006.

