MUFFAKHAM JAH COLLEGE OF ENGINEERING AND TECHNOLOGY

Banjara Hills, Hyderabad, Telangana

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

Object Oriented And System Design Laboratory Manual

Academic Year 2016-2017

Table of Contents

1. Vision of the Institution

To be part of universal human quest for development and progress by contributing high calibre, ethical and socially responsible engineers who meet the global challenge of building modern society in harmony with nature.

2. Mission of the Institution

- To attain excellence in imparting technical education from undergraduate through doctorate levels by adopting coherent and judiciously coordinated curricular and co-curricular programs.
- To foster partnership with industry and government agencies through collaborative research and consultancy.
- To nurture and strengthen auxiliary soft skills for overall development and improved employability in a multi-cultural work space.
- To develop scientific temper and spirit of enquiry in order to harness the latent innovative talents.
- To develop constructive attitude in students towards the task of nation building and empower them to become future leaders
- To nourish the entrepreneurial instincts of the students and hone their business acumen.
- To involve the students and the faculty in solving local community problems through economical and sustainable solutions.

3. Department Vision

To contribute competent computer science professionals to the global talent pool to meet the constantly evolving societal needs.

4. Department Mission

Mentoring students towards a successful professional career in a global environment through quality education and soft skills in order to meet the evolving societal needs.

5. Programme Education Objectives

1. Graduates will demonstrate technical skills and leadership in their chosen fields of employment by solving real time problems using current techniques and tools.

2. Graduates will communicate effectively as individuals or team members and be successful in the local and global cross cultural working environment.

3. Graduates will demonstrate lifelong learning through continuing education and professional development.

4. Graduates will be successful in providing viable and sustainable solutions within societal, professional, environmental and ethical contexts

6. Programme Outcomes

1. **Engineering knowledge:** Apply the knowledge of mathematics, science, engineering fundamentals and an engineering specialization to the solution of complex engineering problems.

2. **Problem analysis:** Identify, formulate, review research literature, and analyze complex engineering problems reaching substantiated conclusions using first principles of mathematics, natural sciences, and engineering sciences.

3. **Design/development of solutions:** Design solutions for complex engineering problems and design system components or processes that meet the specified needs with appropriate consideration for the public health and safety, and the cultural, societal, and environmental considerations.

4. **Conduct investigations of complex problems:** Use research-based knowledge and research methods including design of experiments, analysis and interpretation of data, and synthesis of the information to provide valid conclusions.

5. **Modern tool usage:** Create, select, and apply appropriate techniques, resources, and modern engineering and IT tools including prediction and modeling to complex engineering activities with an understanding of the limitations.

6. **The engineer and society:** Apply reasoning informed by the contextual knowledge to assess societal, health, safety, legal and cultural issues and the consequent responsibilities relevant to the professional engineering practice.

7. **Environment and sustainability:** Understand the impact of the professional engineering solutions in societal and environmental contexts, and demonstrate the knowledge of, and need for sustainable development.

8. **Ethics:** Apply ethical principles and commit to professional ethics and responsibilities and norms of the engineering practice.

9. **Individual and team work:** Function effectively as an individual, and as a member or leader in diverse teams, and in multidisciplinary settings.

10. **Communication:** Communicate effectively on complex engineering activities with the engineering community and with society at large, such as, being able to comprehend and write effective reports and design documentation, make effective presentations, and give and receive clear instructions.

11. **Project management and finance:** Demonstrate knowledge and understanding of the engineering and management principles and apply these to one's own work, as a member and leader in a team, to manage projects and in multidisciplinary environments.

12. **Lifelong learning:** Recognize the need for, and have the preparation and ability to engage in independent and lifelong learning in the broadest context of technological change.

7. Programme Specific Outcomes

The graduates will be able to:

PSO1: Demonstrate understanding of the principles and working of the hardware and software aspects of computer systems.

PSO2: Use professional engineering practices, strategies and tactics for the development, operation and maintenance of software

PSO3: Provide effective and efficient real time solutions using acquired knowledge in various domains.

8. Introduction

OVERVIEW OF UNIFIED MODELLING LANGUAGE

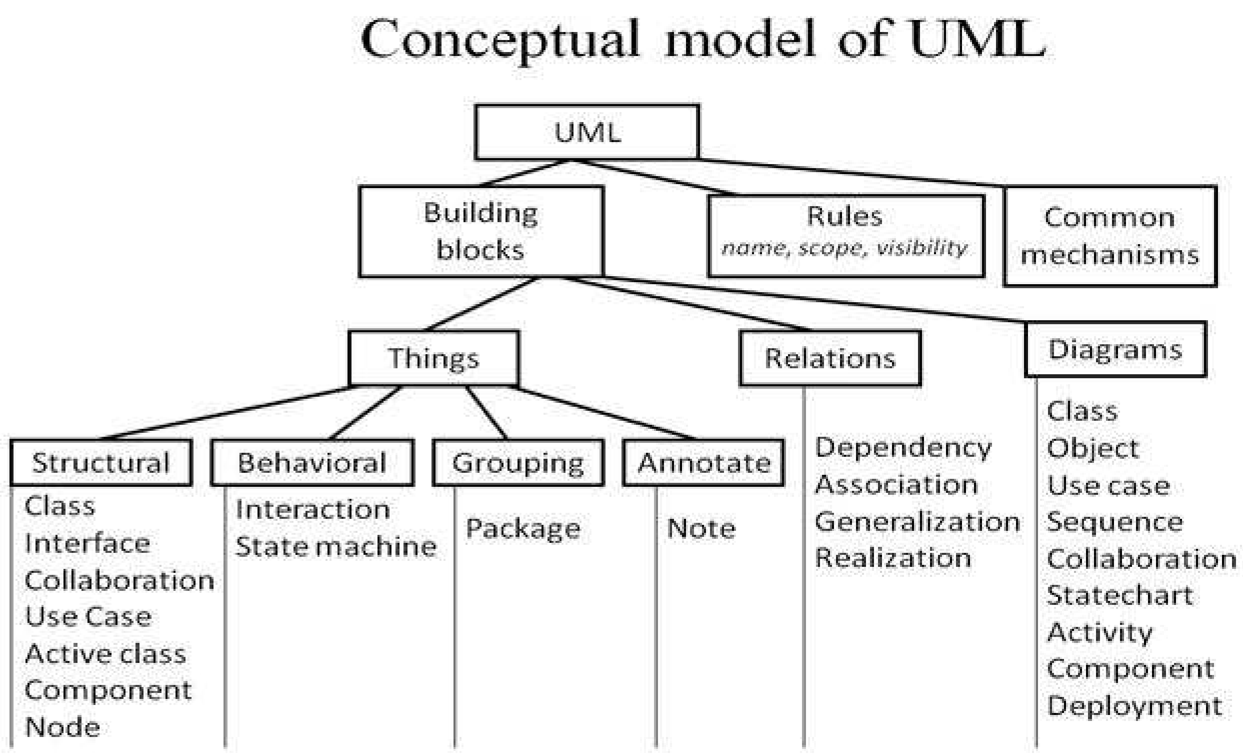

Figure 1: Shows the Conceptual model of UML

The UML Conceptual model of UML is shown in Figure 1. The building blocks of UML are:

- Things
- Relationships
- Diagrams

1. **Things:** Things are the most important building blocks of UML. Things can be:
   - Structural
   - Behavioral
   - Grouping
   - Annotational

Structural things:

The **Structural things** define the static part of the model. They represent physical and conceptual elements. Following are the brief descriptions of the structural things.

Class: Class represents set of objects having similar responsibilities, which is shown in Figure 2.

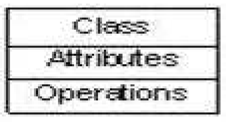

Figure 2: Compartments of Class

Interface: Interface defines a set of operations which specify the responsibility of a class, shown in Figure 3.

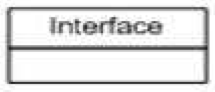

Figure 3: Interface symbol that has set of operations

Collaboration: Collaboration defines interaction between elements, shown in Figure 4.

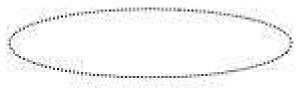

Figure 4: Collaboration Symbol

Use case: Use case represents a set of actions performed by a system for a specific goal. shown in Figure 5.

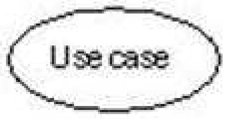

Figure 5: Use case Symbol

Component: Component describes physical part of a system. shown in Figure 6.

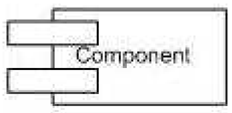

Figure 6: Component Symbol

Node: A node can be defined as a physical element that exists at run time. shown in Figure 7.

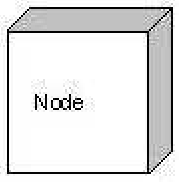

Figure 7: Node Symbol

Behavioral things: A behavioral thing consists of the dynamic parts of UML models. Following are the behavioral things:

Interaction: Interaction is defined as a behavior that consists of a group of messages exchanged among elements to accomplish a specific task shown in Figure 8.

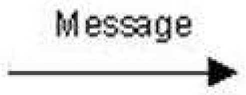

Figure 8: Interaction Symbol

State machine: State machine is useful when the state of an object in its life cycle is important. It defines the sequence of states an object goes through in response to events. Events are external factors responsible for state change. shown in Figure 9.

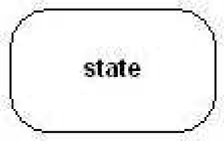

Figure 9: State Machine Symbol

Grouping things: Grouping things can be defined as a mechanism to group elements of a UML model together. There is only one grouping thing available:

Package: Package is the only one grouping thing available for gathering structural and behavioral things. shown in Figure 10.

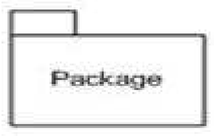

Figure 10: Package Symbol

Annotational things: Annotational things can be defined as a mechanism to capture remarks, descriptions, and comments of UML model elements. Note is the only one Annotational thing available.

Note: A note is used to render comments, constraints etc of an UML element. shown in Figure 11.

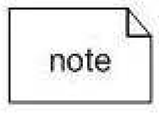

Figure 11: Note Symbol

2. **Relationship :**

   Relationship is another most important building block of UML. It shows how elements are associated with each other and this association describes the functionality of an application. There are four kinds of relationships available. **Dependency:** Dependency is a relationship between two things in which change in one element also affects the other one. shown in Figure 12.

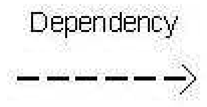

Figure 12: Relationship

Association: Association is basically a set of links that connects elements of an UML model. It also describes how many objects are taking part in that relationship, shown in Figure 13.

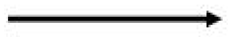

Figure 13: Association Symbol

Generalization: Generalization can be defined as a relationship which connects a specialized element with a generalized element. It basically describes inheritance relationship in the world of objects. shown in Figure 14.

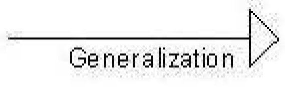

Figure 14: Generalization Symbol

Realization: Realization can be defined as a relationship in which two elements are connected. One element describes some responsibility which is not implemented and the other one implements them. This relationship exists in case of interfaces. shown in Figure 15.

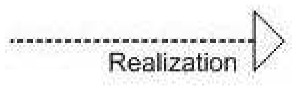

Figure 15: Realization Symbol

3. **UML Diagrams:**
   UML diagrams are the ultimate output of the entire discussion. All the elements, relationships are used to make a complete UML diagram and the diagram represents a system. The visual effect of the UML diagram is the most important part of the entire process. All the other elements are used to make it a complete one. UML includes the following nine diagrams and the details are described in the following chapters.

   - Class diagram

- Object diagram
- Use case diagram
- Sequence diagram
- Collaboration diagram
- Activity diagram
- Statechart diagram
- Deployment diagram
- Component diagram

Different diagrams are used for different type of UML modeling. There are three important type of UML modelings: Structural modeling: Structural modeling captures the static features of a system. They consist of the followings:

- Classes diagrams
- Objects diagrams
- Deployment diagrams
- Package diagrams
- Composite structure diagram
- Component diagram

Structural model represents the framework for the system and this framework is the place where all other components exist. So the class diagram, component diagram and deployment diagrams are the part of structural modeling. They all represent the elements and the mechanism to assemble them. But the structural model never describes the dynamic behavior of the system. Class diagram is the most widely used structural diagram.

Behavioral Modeling: Behavioral model describes the interaction in the system. It represents the interaction among the structural diagrams. Behavioral modeling shows the dynamic nature of the system. They consist of the following:

- Activity diagrams
- Interaction diagrams
- Use case diagrams

All the above show the dynamic sequence of flow in a system.

Architectural Modeling: Architectural model represents the overall framework of the system. It contains both structural and behavioral elements of the system. Architectural model can be defined as the blue print of the entire system. Package diagram comes under architectural modeling.

Structural Things: Graphical notations used in structural things are the most widely used in UML. These are considered as the nouns of UML models. Following are the list of structural things.

- Classes
- Interface
- Collaboration
- Use case
- Active classes
- Components
- Nodes

Class Notation: UML class is represented by the diagram shown in Figure 16. The diagram is divided into four parts.

- The top section is used to name the class.
- The second one is used to show the attributes of the class.
- The third section is used to describe the operations performed by the class.
- The fourth section is optional to show any additional components.

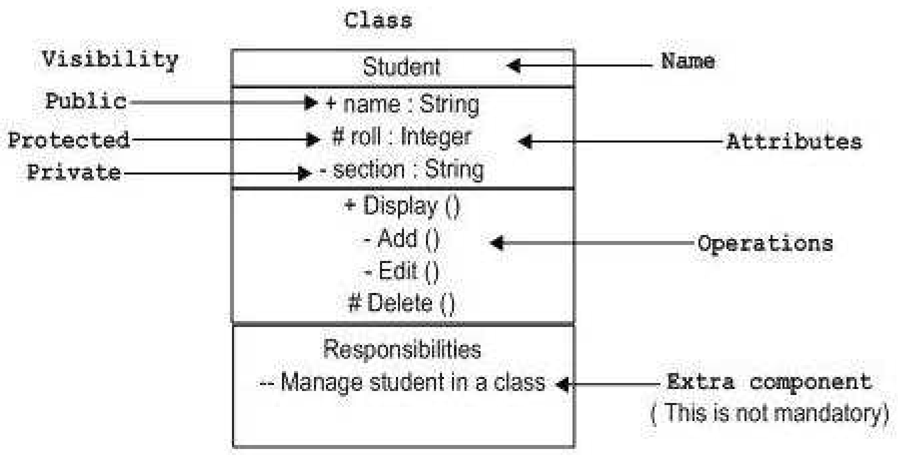

Figure 16: Class Symbol

Classes are used to represent objects. Objects can be anything having properties and responsibility.

Object Notation: The object is represented in the same way as the class. The only difference is the name which is underlined as shown in Figure 17.

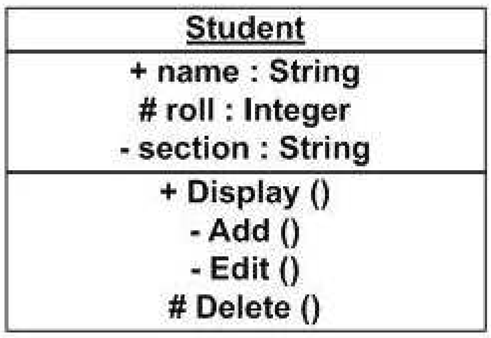

Figure 17: Object Symbol where Student object is underlined

As object is the actual implementation of a class which is known as the instance of a class. So it has the same usage as the class.

Interface Notation: Interface is represented by a circle as shown in Figure 18. It has a name which is generally written below the circle.

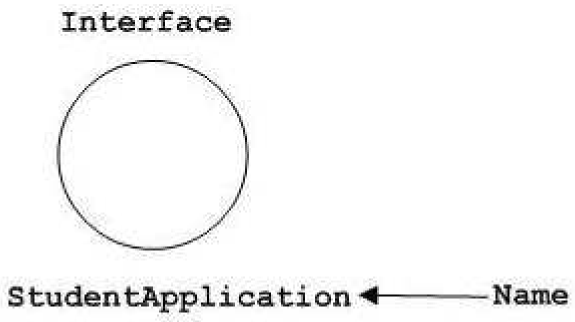

Figure 18: Interaface Symbol

Interface is used to describe functionality without implementation. Interface is the just like a template where you define different functions not the implementation. When a class implements the interface it also implements the functionality as per the requirement.

Collaboration Notation: Collaboration is represented by a dotted eclipse as shown in Figure 19. It has a name written inside the eclipse.

Collaboration represents responsibilities. Generally responsibilities are in a group.

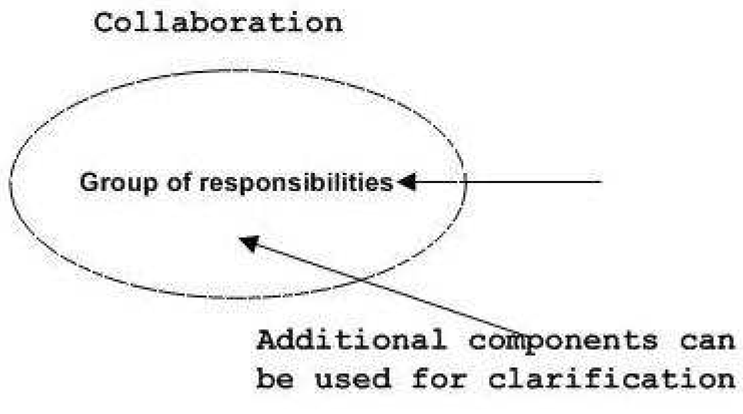
Collaboration
Group of responsibilities
Additional components can be used for clarification

Figure 19: Collaboration Notation

Use case Notation:

Use case is represented as an eclipse with a name inside it, is shown in Figure 20 . It may contain additional responsibilities.

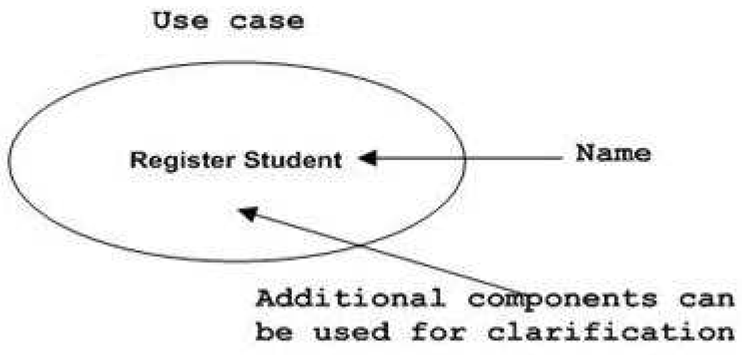

Figure 20: Use case that captures set of activities

Use case is used to capture high level functionalities of a system.

Actor Notation: An actor can be defined as some internal or external entity that interacts with the system. Actor is used in a use case diagram to describe the internal or external entities.

Initial State Notation: Initial state is defined to show the start of a process, shown in Figure 21. This notation is used in almost all diagrams.

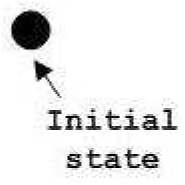

Figure 21: Shows Initial state

The usage of Initial State Notation is to show the starting point of a process.

Final State Notation: Final state is used to show the end of a process, shown in Figure 22. This notation is also used in almost all diagrams to describe the end.

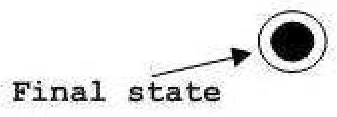

Figure 22: Final state notation

The usage of Final State Notation is to show the termination point of a process.

Active class Notation: Active class looks similar to a class with a solid border, shown in Figure 23. Active class is generally used to describe concurrent behaviour of a system.

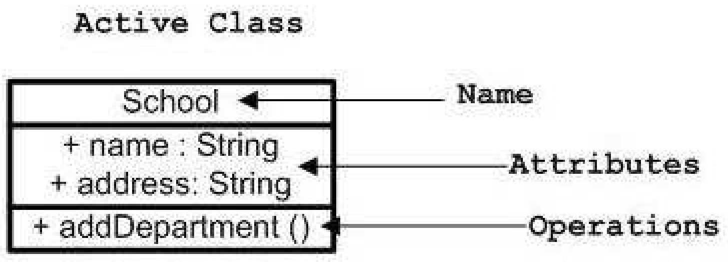

Figure 23: Active class with outline bolded

Active class is used to represent concurrency in a system.

Component Notation: A component in UML is shown as below with a name inside, shown in Figure 24. Additional elements can be added wherever required.

Component is used to represent any part of a system for which UML diagrams are made.

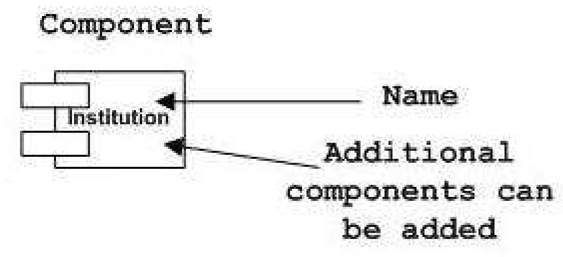

Figure 24: Component Notation

Node Notation: A node in UML is represented by a square box as shown with a name, shown in Figure 25. A node represents a physical component of the system.

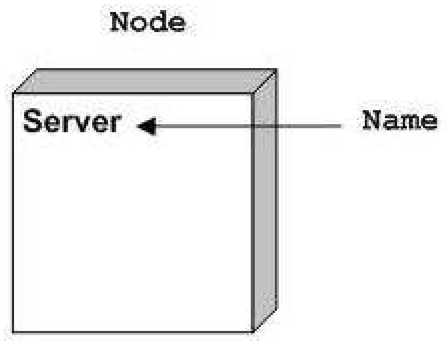

Figure 25: Node Notation

Node is used to represent physical part of a system like server, network etc.

Behavioural Things: Dynamic parts are one of the most important elements in UML. UML has a set of powerful features to represent the dynamic part of software and non software systems. These features include interactions and state machines. Interactions can be of two types:

- Sequential (Represented by sequence diagram)
- Collaborative (Represented by collaboration diagram)

Interaction Notation: Interaction is basically message exchange between two UML components, shown in Figure 26. The following diagram represents different notations used in an interaction.

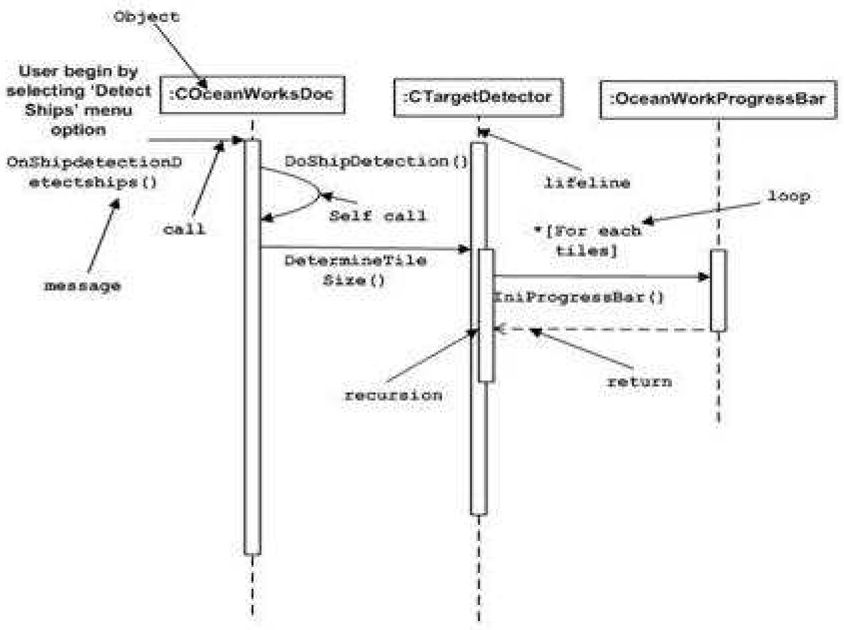

Figure 26: The interaction diagram

Interaction is used to represent communication among the components of a system.

State machine Notation: State machine describes the different states of a component in its life cycle, shown in Figure 27. The notations are described in the following diagram.

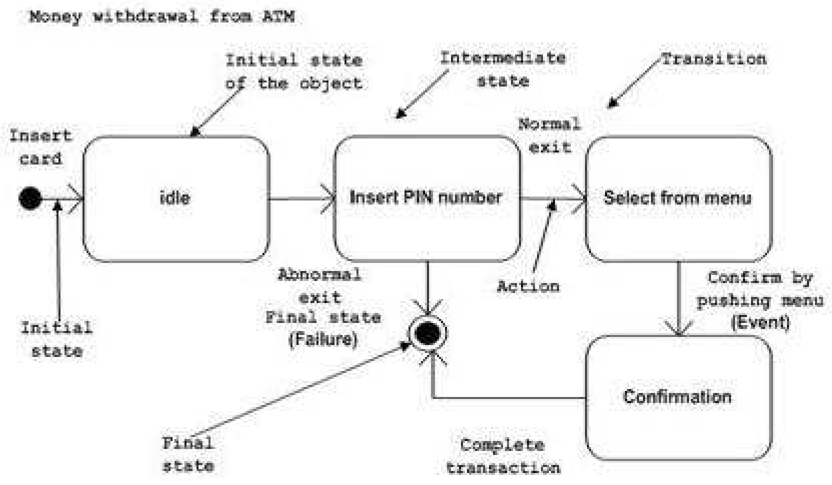

Figure 27: State Machine flows and its events

State machine is used to describe different states of a system component. The state can be active, idle or any other depending upon the situation.

Grouping Things: Organizing the UML models are one of the most important aspects of the design. In UML there is only one element available for grouping and that is package.

Package Notation: Package notation is shown below and this is used to wrap the components of a system. shown in Figure 28.

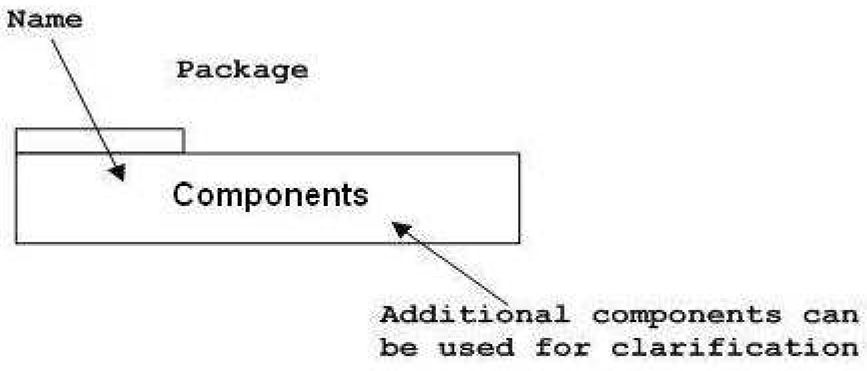

Figure 28: Package for grouping things

Annotational Things: In any diagram explanation of different elements and their functionalities are very important. So UML has notes notation to support this requirement.

Note Notation: This notation is shown below and they are used to provide necessary information of a system. shown in Figure 29

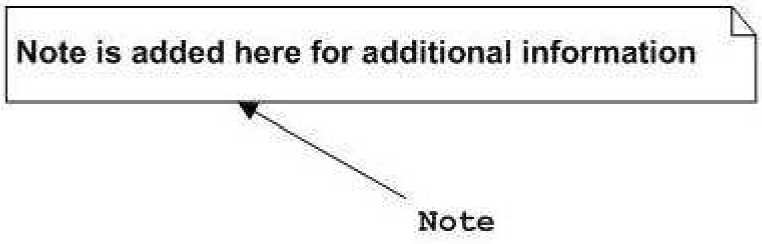

Figure 29: Note that represents Comments

Relationships A model is not complete unless the relationships between elements are described properly. The Relationship gives a proper meaning to an UML model. Following are the different types of relationships available in UML.

- Dependency
- Association
- Generalization
- Extensibility

Dependency Notation: Dependency is an important aspect in UML elements. It describes the dependent elements and the direction of dependency. Dependency is represented by a dotted arrow as shown shown in Figure 30. The arrow head represents the independent element and the other end the dependent element.

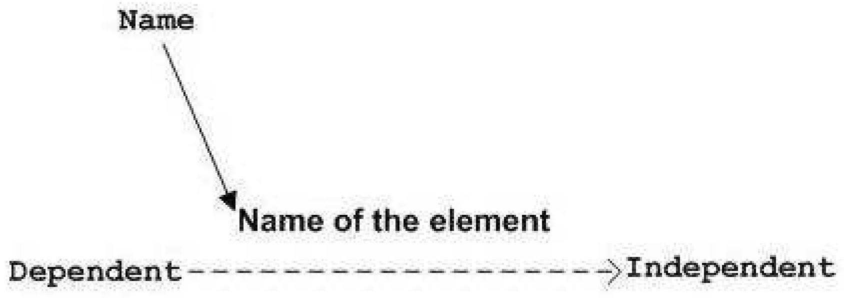

Figure 30: Dependency Notation

Dependency is used to represent dependency between two elements of a system.

Association Notation: Association describes how the elements in an UML diagram are associated, shown in Figure 31. In simple word it describes how many elements are taking part in an interaction. Association is represented by a dotted line with (without) arrows on both sides. The two ends represent two associated elements as shown below. The multiplicity is also mentioned at the ends (1, * etc) to show how many objects are associated.

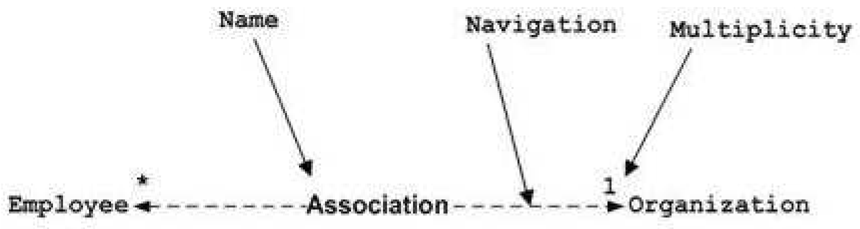

Figure 31: Association and multiplicity constraint (The line must be a Solid line)

Association is used to represent the relationship between two elements of a system.

Generalization Notation: Generalization describes the inheritance relationship of the object oriented world, shown in Figure 32. It is parent and child relationship. Generalization is represented by an arrow with hollow arrow head as shown below. One end represents the parent element and the other end child element.

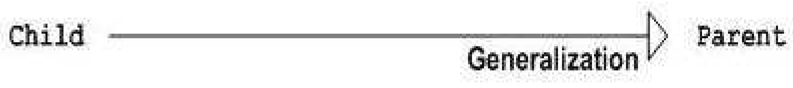

Figure 32: Generalization Notation

Generalization is used to describe parent-child relationship of two elements of a system.

Extensibility Notation: All the languages (programming or modeling) have some mechanism to extend its capabilities like syntax, semantics etc, shown in Figure 33. UML is also having the following mechanisms to provide extensibility features.

- Stereotypes (Represents new elements)
- Tagged values (Represents new attributes)
- Constraints (Represents the boundaries)

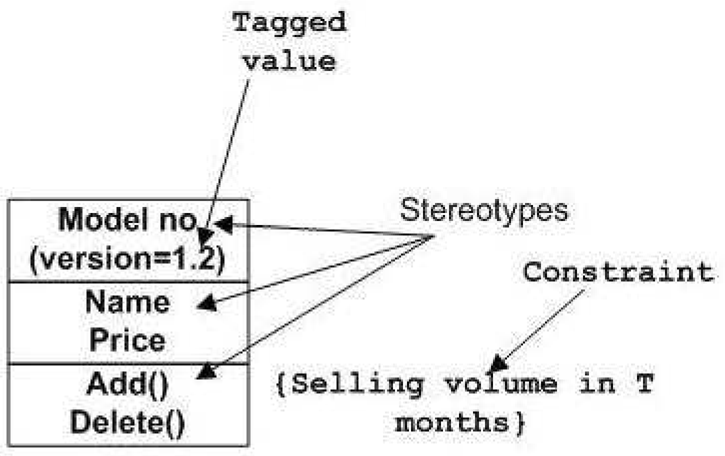

Figure 33: shows Stereotype, Tagged values, constraint

Extensibility notations are used to enhance the power of the language. It is basically additional elements used to represent some extra behaviour of the system. These extra behaviours are not covered by the standard available notations.

Class diagram

The class diagram is a static diagram. It represents the static view of an application, shown in Figure 34. 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.

Purpose:
The purpose of the class diagram is to model the static view of an application. The class diagrams are the only diagrams which can be directly mapped with object oriented languages and thus widely used at the time of construction.

The UML diagrams like activity diagram, sequence diagram can only give the sequence flow of the application but class diagram is a bit different. So it is the most popular UML diagram in the coder community. So the purpose of the class diagram can be summarized as:

- Analysis and design of the static view of an application.
- Describe responsibilities of a system.
- Base for component and deployment diagrams.
- Forward and reverse engineering.

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 is an abstract class and it has two concrete classes (inheritance relationship) *SpecialOrder* and *NormalOrder*.
- 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:

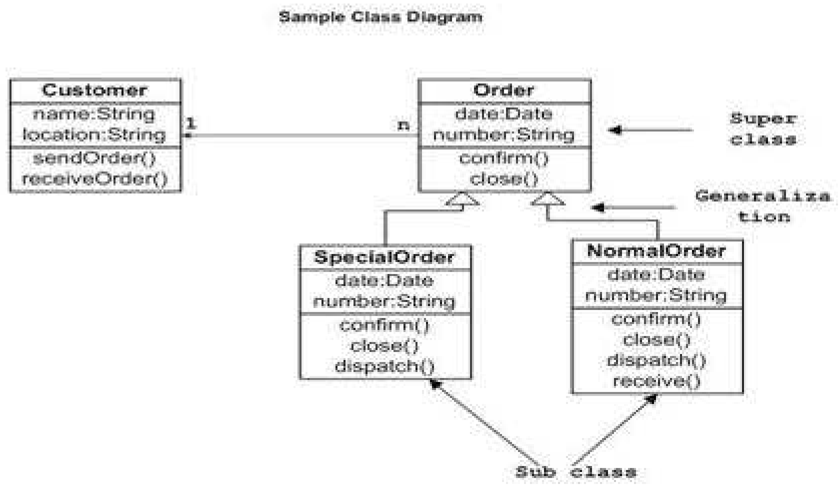

Figure 34: Shows the Class diagram

Component Diagram

Component diagram is a special kind of diagram in UML, shown in Figure 35. The purpose is also different from all other diagrams discussed so far. It does not describe the functionality of the system but it describes the components used to make those functionalities.

So from that point component diagrams are used to visualize the physical components in a system. These components are libraries, packages, files etc.

Component diagrams can also be described as a static implementation view of a system. Static implementation represents the organization of the components at a particular moment.

A single component diagram cannot represent the entire system but a collection of diagrams are used to represent the whole. So the purpose of the component diagram can be summarized as:

- Visualize the components of a system.
- Construct executables by using forward and reverse engineering.
- Describe the organization and relationships of the components.

Component diagrams are used to describe the physical artifacts of a system. This artifact includes files, executables, libraries etc.

So the purpose of this diagram is different, Component diagrams are used during the implementation phase of an application. But it is prepared well in advance to visualize the implementation details. Initially the system is designed using different UML diagrams and then when the artifacts are ready component diagrams are used to get an idea of the implementation.

This diagram is very important because without it the application cannot be implemented efficiently. A well prepared component diagram is also important for other aspects like application performance, maintenance etc.

So before drawing a component diagram the following artifacts are to be identified clearly:

- Files used in the system.
- Libraries and other artifacts relevant to the application.
- Relationships among the artifacts.

Now after identifying the artifacts the following points needs to be followed:

- Use a meaningful name to identify the component for which the diagram is to be drawn.
- Prepare a mental layout before producing using tools.
- Use notes for clarifying important points.

The following is a component diagram for order management system. Here the artifacts are files. So the diagram shows the files in the application and their relationships. In actual the component diagram also contains dlls, libraries, folders etc.

In the following diagram four files are identified and their relationships are produced. Component diagram cannot be matched directly with other UML diagrams discussed so far. Because it is drawn for completely different purpose.

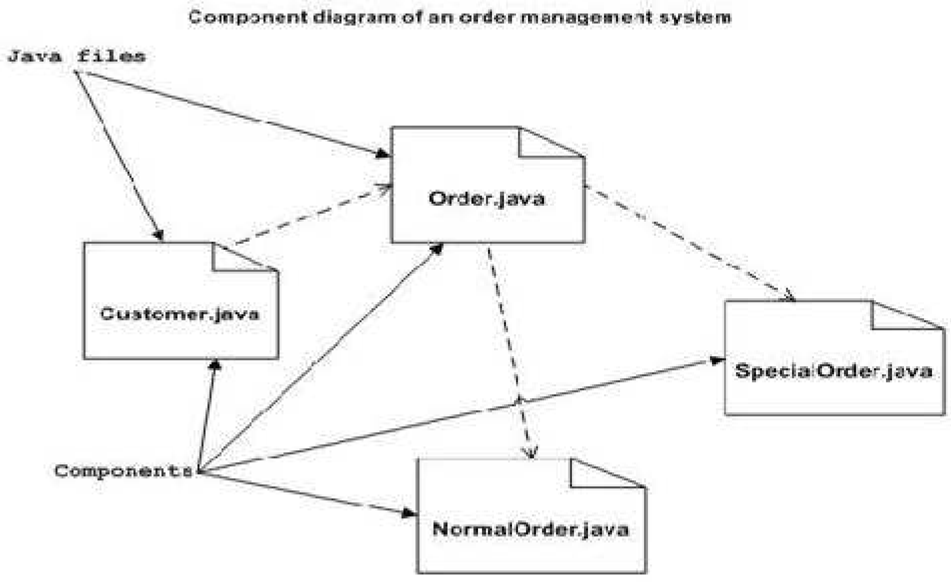

Figure 35: Component Diagrams representation using source files

These diagrams show the physical components of a system. To clarify it, we can say that component diagrams describe the organization of the components in a system.

Organization can be further described as the location of the components in a system. These components are organized in a special way to meet the system requirements.

As we have already discussed those components are libraries, files, executables etc. Now before implementing the application these components are to be organized. This component organization is also designed separately as a part of project execution.

Component diagrams are very important from implementation perspective. So the implementation team of an application should have a proper knowledge of the component details. Now the usage of component diagrams can be described as:

- Model the components of a system.
- Model database schema.
- Model executables of an application.
- Model system's source code.

Deployment Diagram

The name Deployment itself describes the purpose of the diagram, shown in Figure 36. 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 can be described as:

- Visualize hardware topology of a system.
- Describe the hardware components used to deploy software components.
- Describe runtime processing nodes.

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 hardwares 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.

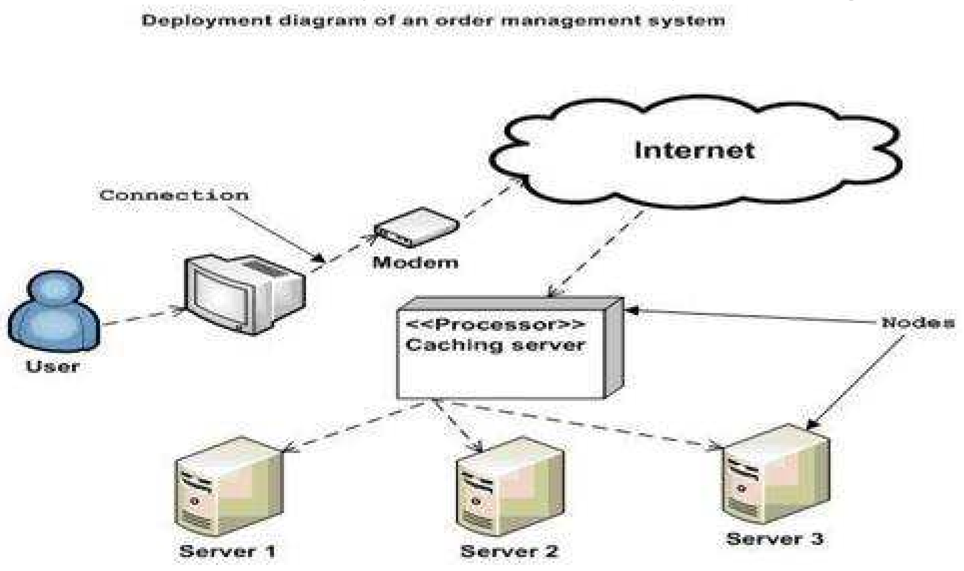

Figure 36: Deployment diagram shows hardware configuration

Deployment diagrams are mainly used by system engineers. These diagrams are used to describe the physical components (hardware), their distribution and association.

To clarify it in details we can visualize deployment diagrams as the hardware components/nodes on which software components reside.

Software applications are developed to model complex business processes. Only efficient software applications are not sufficient to meet business requirements. Business requirements can be described as to support increasing number of users, quick response time etc.

To meet these types of requirements hardware components should be designed efficiently and in a cost effective way.

Now a day's software applications are very complex in nature. Software applications can be stand alone, web based, distributed, mainframe based and many more. So it is very important to design the hardware components efficiently. So the usage of deployment diagrams can be described as follows:

- To model the hardware topology of a system.
- To model embedded system.
- To model hardware details for a client/server system.
- To model hardware details of a distributed application.
- Forward and reverse engineering.

Use case Diagrams

Use case diagrams are used to gather the requirements of a system including internal and external influences, shown in Figure 37. 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.

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, SpecialOrder and NormalOrder) and one actor which is customer. The *SpecialOrder* and *NormalOrder* 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.

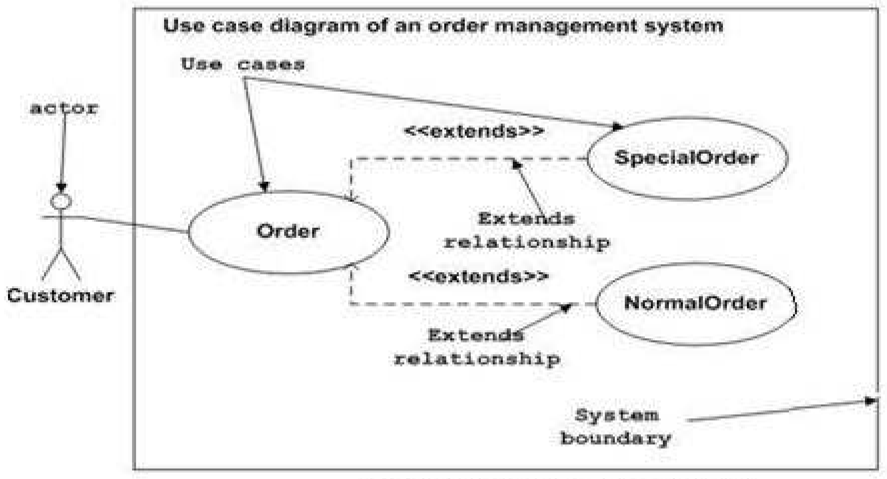

Figure: Sample Use Case diagram

Figure 37: Shows use-case diagram

Use case diagrams specify the events of a system and their flows. But use case diagram never describes how they are implemented. Use case diagram can be imagined as a black box where only the input, output and the function of the black box is known.

These diagrams are used at a very high level of design. Then this high level design is refined again and again to get a complete and practical picture of the system. A well structured use case also describes the pre condition, post condition, exceptions. And these extra elements are used to make test cases when performing the testing. Although the use cases are not a good candidate for forward and reverse engineering but still they are used in a slight different way to make forward and reverse engineering. And the same is true for reverse engineering. Still use case diagram is used differently to make it a candidate for reverse engineering.

In forward engineering use case diagrams are used to make test cases and in reverse engineering use cases are used to prepare the requirement details from the existing application. So the following are the places where use case diagrams are used:

- Requirement analysis and high level design.
- Model the context of a system.
- Reverse engineering.
- Forward engineering.

Interaction Diagrams

The purpose of interaction diagrams is to visualize the interactive behaviour of the system, shown in Figure 38. Now visualizing interaction is a difficult task. So the solution is to use different types of models to capture the different aspects of the interaction.

That is why sequence and collaboration diagrams are used to capture dynamic nature but from a different angle. So the purpose of interaction diagram can be described as:

- To capture dynamic behaviour of a system.
- To describe the message flow in the system.
- To describe structural organization of the objects.
- To describe interaction among objects.

We have two types of interaction diagrams in UML. One is sequence diagram and the other is a collaboration diagram. The sequence diagram captures the time sequence of message flow from one object to another and the collaboration diagram describes the organization of objects in a system taking part in the message flow. So the following things are to identified clearly before drawing the interaction diagram:

- Objects taking part in the interaction.
- Message flows among the objects.
- The sequence in which the messages are flowing.
- Object organization.

Following are two interaction diagrams modeling order management system. The first diagram is a sequence diagram and the second is a collaboration diagram.

The Sequence Diagram:

The sequence diagram is having four objects (Customer, Order, SpecialOrder and NormalOrder). The following diagram has shown the message sequence for *SpecialOrder* object and the same can be used in case of *NormalOrder* object. Now it is important to understand the time sequence of message flows. The message flow is nothing but a method call of an object.

The first call is *sendOrder ()* which is a method of *Order* object. The next call is *confirm ()* which is a method of *SpecialOrder* object and the last call is *Dispatch ()* which is a method of *SpecialOrder* object. So here the diagram is mainly describing the method calls from one object to another and this is also the actual scenario when the system is running.

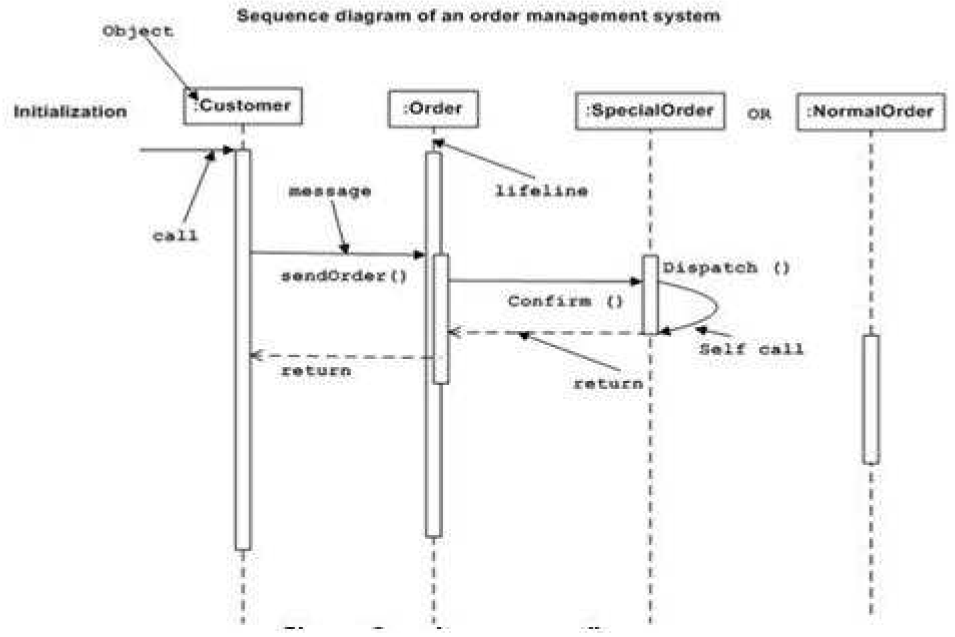

Figure 38: Shows Sequence diagram

Collaboration Diagram:
The second interaction diagram is collaboration diagram. It shows the object organization as shown below. Here in collaboration diagram the method call sequence is indicated by some numbering technique as shown below. The number indicates how the methods are called one after another. We have taken the same order management system to describe the collaboration diagram.

The method calls are similar to that of a sequence diagram. But the difference is that the sequence diagram does not describe the object organization where as the collaboration diagram shows the object organization. Now to choose between these two diagrams the main emphasis is given on the type of requirement. If the time sequence is important then sequence diagram is used and if organization is required then collaboration diagram is used.

The main purpose of both the diagrams are similar as they are used to capture the dynamic behaviour of a system. But the specific purposes are more important to clarify and understood. Sequence diagrams are used to capture the order of messages flowing from one object to another. And the collaboration diagrams are used to describe the structural organizations of the objects taking part in the interaction. A single diagram is not sufficient to describe the dynamic aspect of an entire system so a set of diagrams are used to capture is as a whole.

The interaction diagrams are used when we want to understand the message flow and the structural organization. Now message flow means the sequence of control flow from one object to another and structural organization means the visual organization of the elements in a system.

In a brief the following are the usages of interaction diagrams:

- To model flow of control by time sequence.
- To model flow of control by structural organizations.
- For forward engineering.
- For reverse engineering.

State Chart Diagram Statechart diagram is one of the five UML diagrams used to model dynamic nature of a system, shown in Figure 39. 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.

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

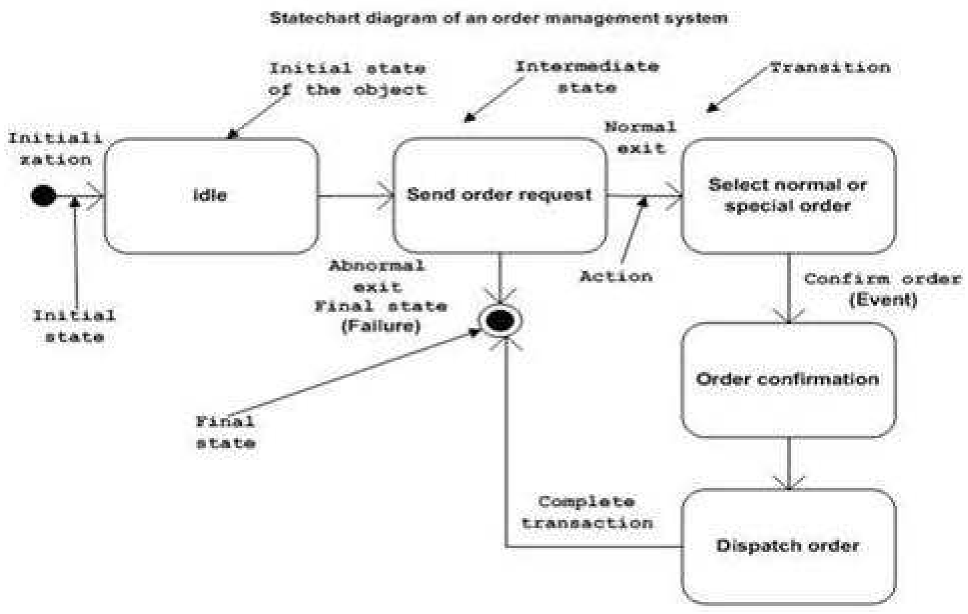

Figure 39: Shows State chart diagram

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.

Statechart diagram defines the states of a component and these state changes are dynamic in nature. So its specific purpose is to define state changes triggered by events. Events are internal or external factors influencing the system.

Statechart diagrams are used to model states and also events operating on the system. When implementing a system it is very important to clarify different states of an object during its life time and statechart diagrams are used for this purpose. When these states and events are identified they are used to model it and these models are used during implementation of the system.

If we look into the practical implementation of Statechart diagram then it is mainly used to analyze the object states influenced by events. This analysis is helpful to understand the system behaviour during its execution. So the main usages can be described as:

- To model object states of a system.
- To model reactive system. Reactive system consists of reactive objects.
- To identify events responsible for state changes.
- Forward and reverse engineering.

Activity Diagram
Activity diagram is another important diagram in UML to describe dynamic aspects of the system, shown in Figure 40. Activity diagram is basically a flow chart to represent the

flow form 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 deals with all type 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 some time considered as the flow chart. Although the diagrams looks like a flow chart but it is not. It shows different flow 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.

Activity diagrams are mainly used as a flow chart consists of activities performed by the system. But activity diagram 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 diagram. 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.

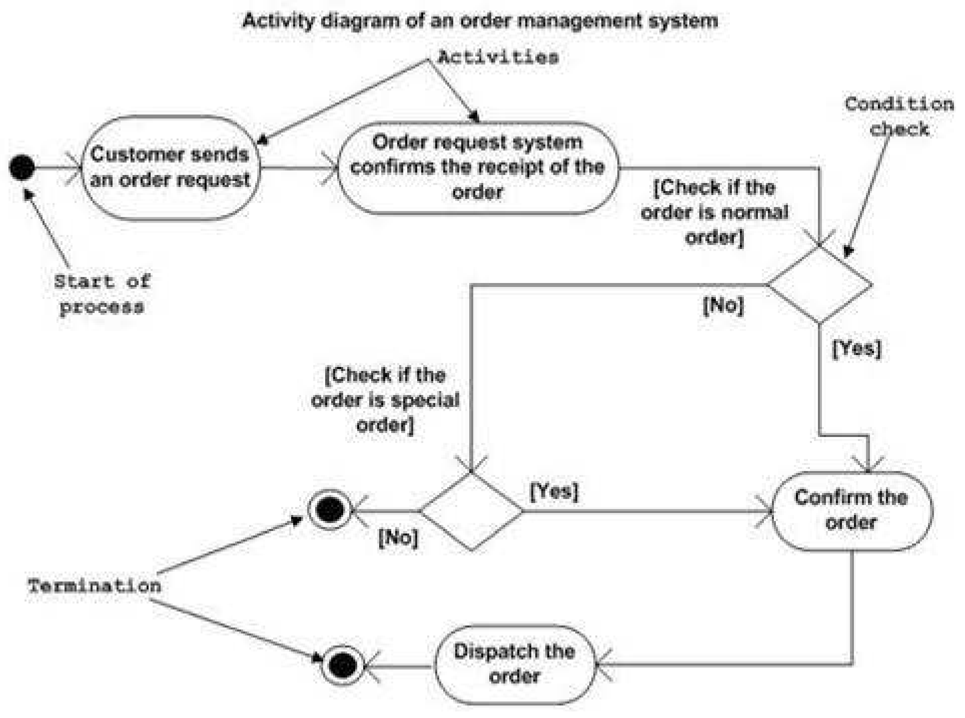
Activity diagram of an order management system
Activities
Condition check
Customer sends an order request
Order request system confirms the receipt of the order
[Check if the order is normal order]
Start of process
[No]
[Yes]
[Check if the order is special order]
[Yes]
[No]
Confirm the order
Termination
Dispatch the order

Figure 40: Shows Activity diagrams

The activity diagram is suitable for modeling the activity flow of the system. An application can have multiple systems. Activity diagram also captures these systems and describes flow from one system to another. This specific usage is not available in other diagrams. These systems can be database, external queues or any other system.

Now we will look into the practical applications of the activity diagram. From the above discussion it is clear that an activity diagram is drawn from a very high level. So it gives high level view of a system. This high level view is mainly for business users or any other person who is not a technical person.

This diagram is used to model the activities which are nothing but business requirements. So the diagram has more impact on business understanding rather implementation details. Following are the main usages of activity diagram:

- Modeling work flow by using activities.
- Modeling business requirements.
- High level understanding of the system's functionalities.
- Investigate business requirements at a later stage.

Laboratory Objective

The aim of Object Oriented System Development laboratory is to provide a foundational base for Modeling the Software system using the scientific approach that aids for Visualizing, Specifying, Constructing and Documenting the minor and major artifacts of the system. It is made by allocating the CASE Studies to the students based on their interest to Understand, Analyze, and then Design the Software System using Unified Modeling Language (UML) diagrams systematically one after the other which are logically connected and later converted to partial programming Code. These diagrams are created through the use of IBM Rational Rose software.

The UML Diagrams for each of the Case Study selected by the Students involves the following diagrams which include:

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

At the end of the lab course the student is equipped with a basic understating, and practical orientation that demonstrates the developed Case study working in real scenario.

9. Sample Case Study for Cab Management System

9..1 Class Diagram

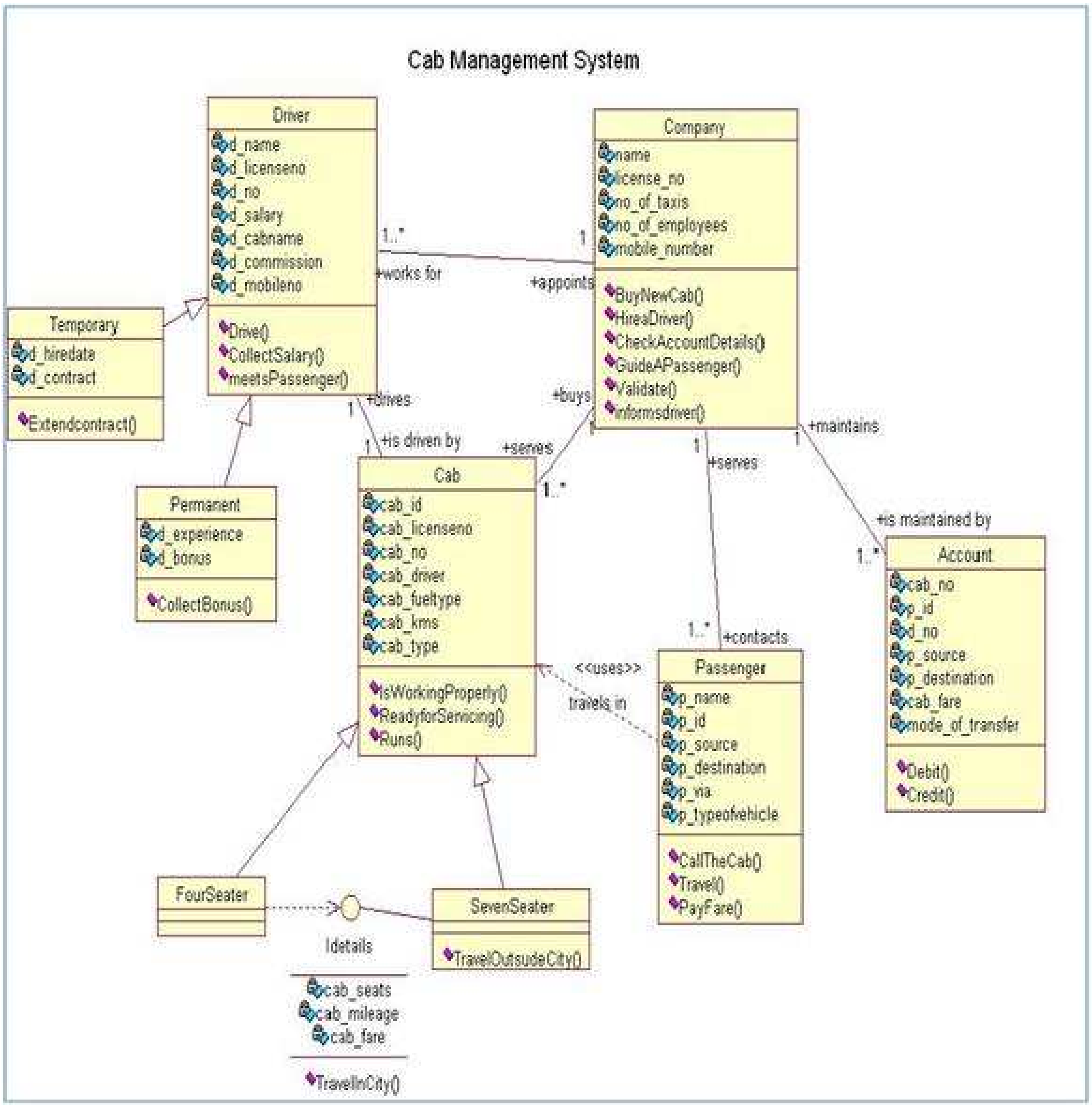

Figure 41: Class diagram for Cab Management System

9..2 Use Case Diagram

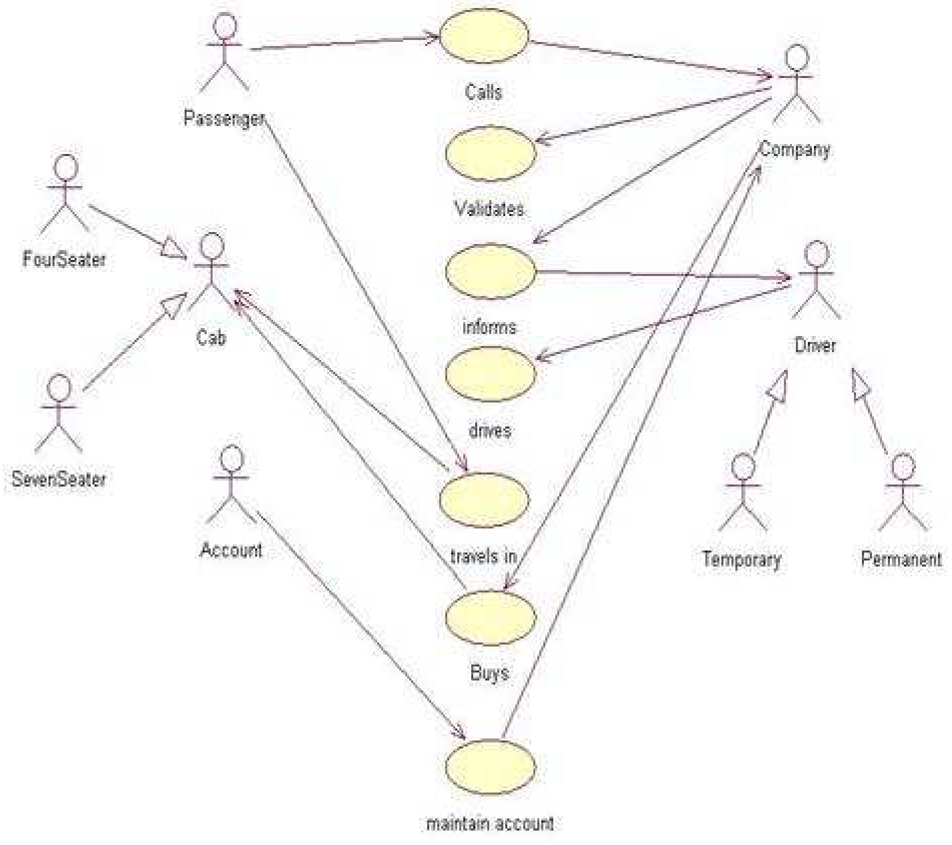

Figure 42: Use case diagram for Cab Management System

9..3 Sequence Diagram

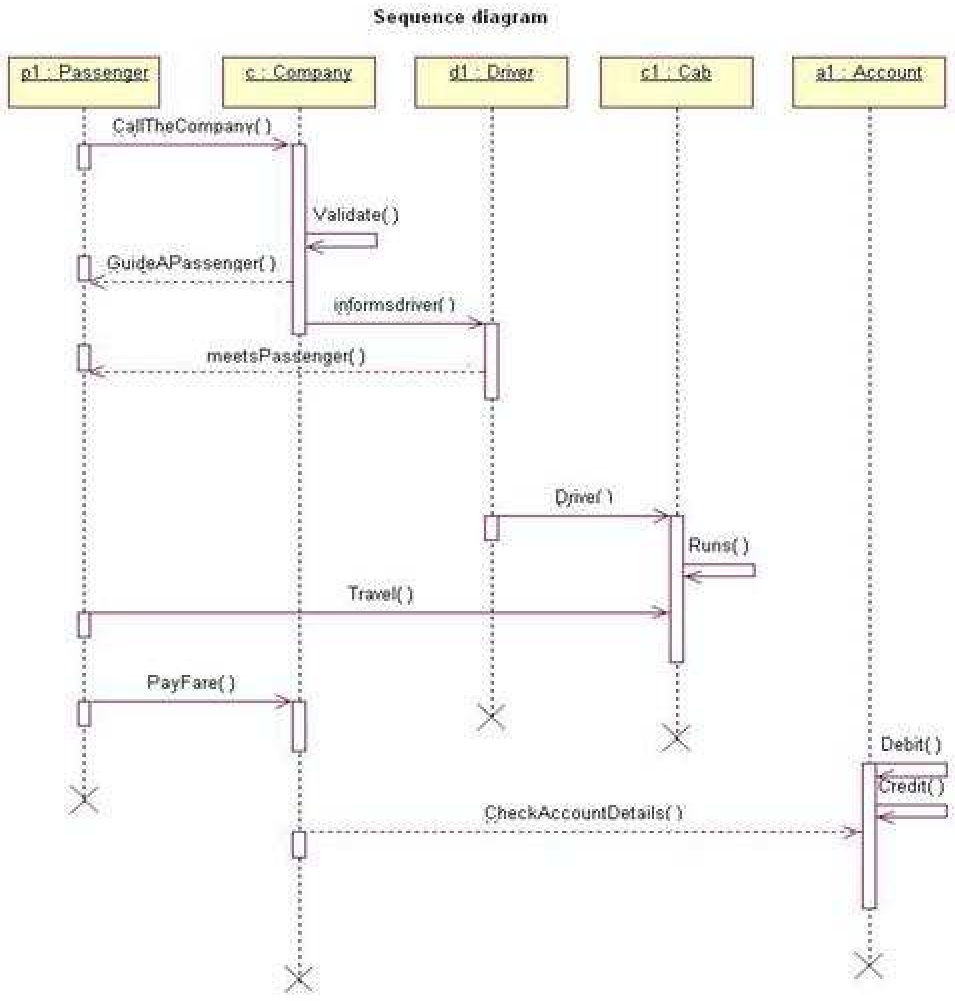

Figure 43: Sequence diagram for Cab Management System

9..4 Collaboration Diagram

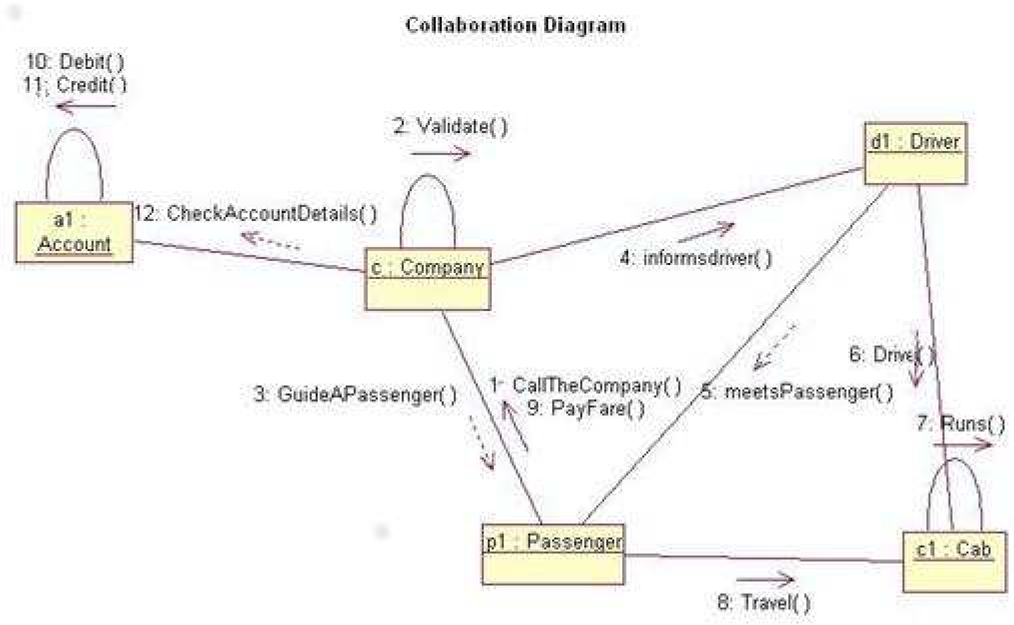

Figure 44: Collaboration of diagram for Cab Management System

9..5 Activity Diagram with Swimlane

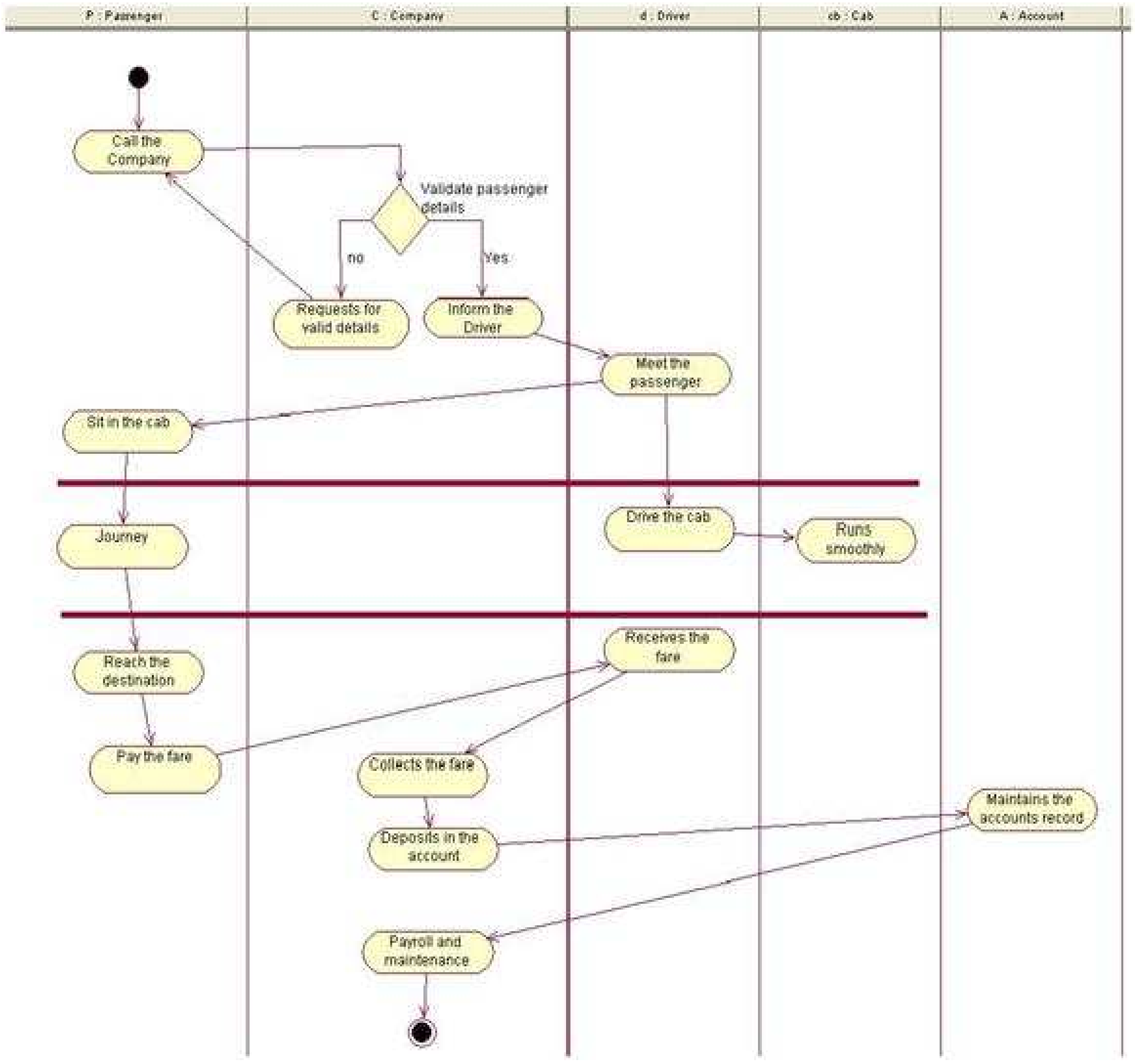

Figure 45: Activity diagram with swimlane for Cab Management System

9..6 State Chart Diagrams for Cab Management System

Passenger

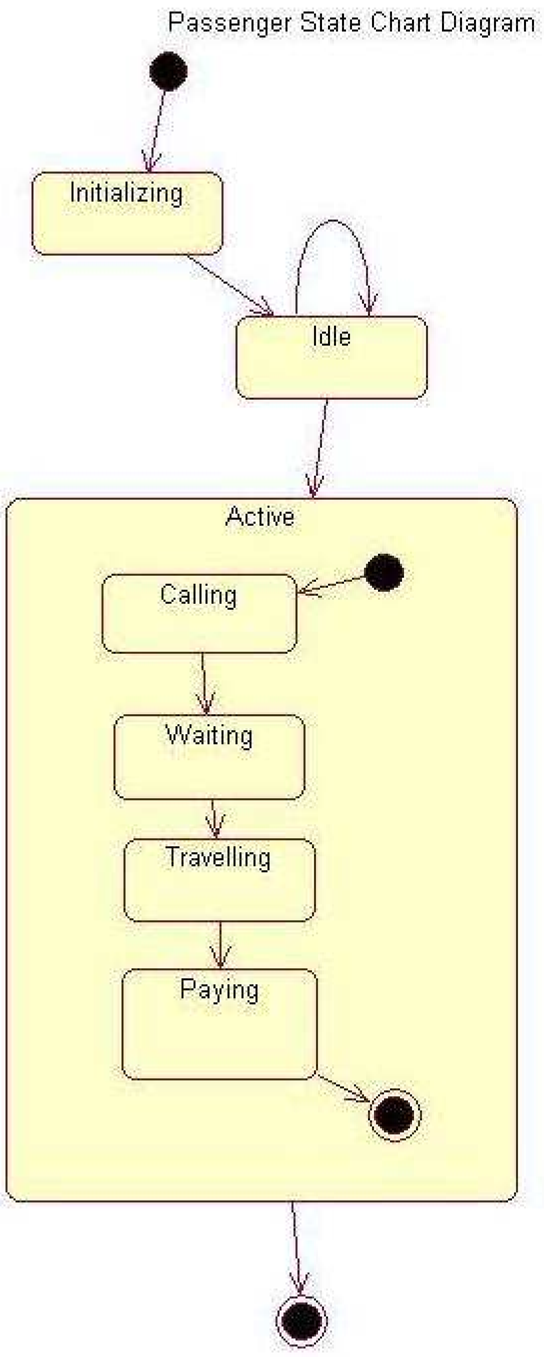

Figure 46: State diagram for Passenger

Company State chart diagram

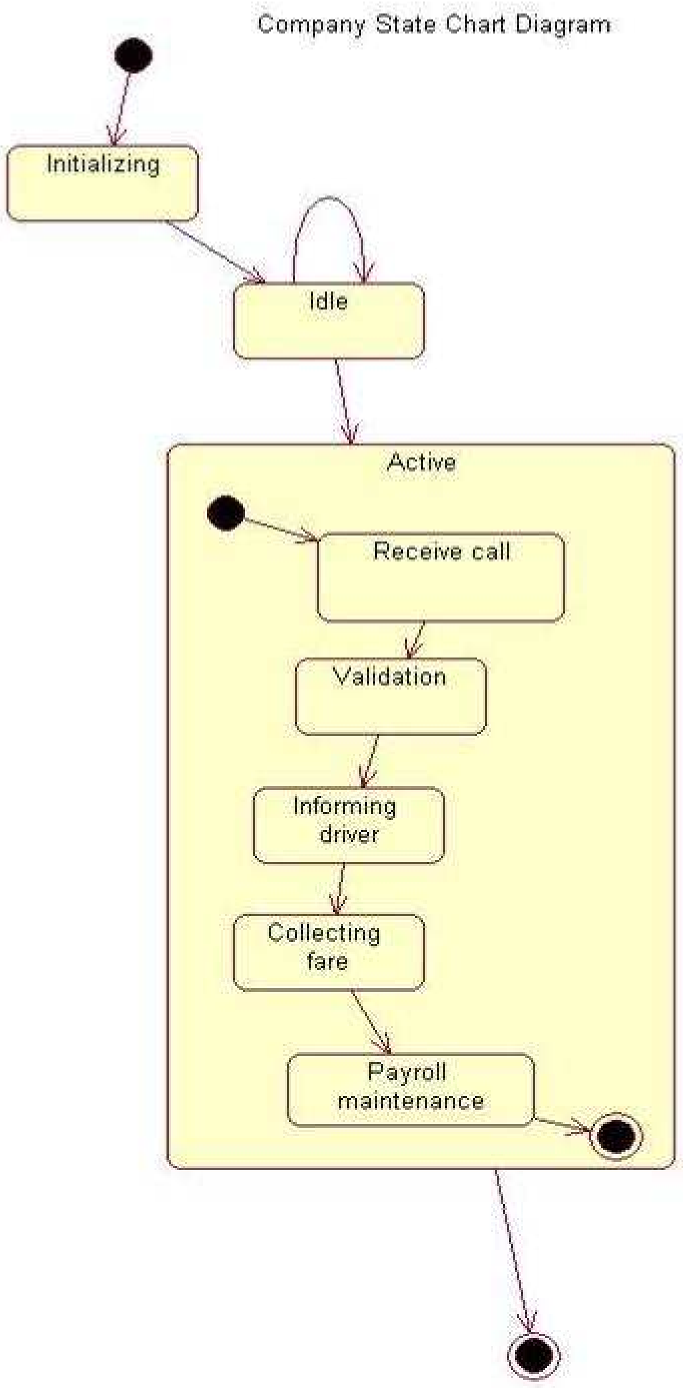

Figure 47: Company State chart diagram

Driver State chart diagram

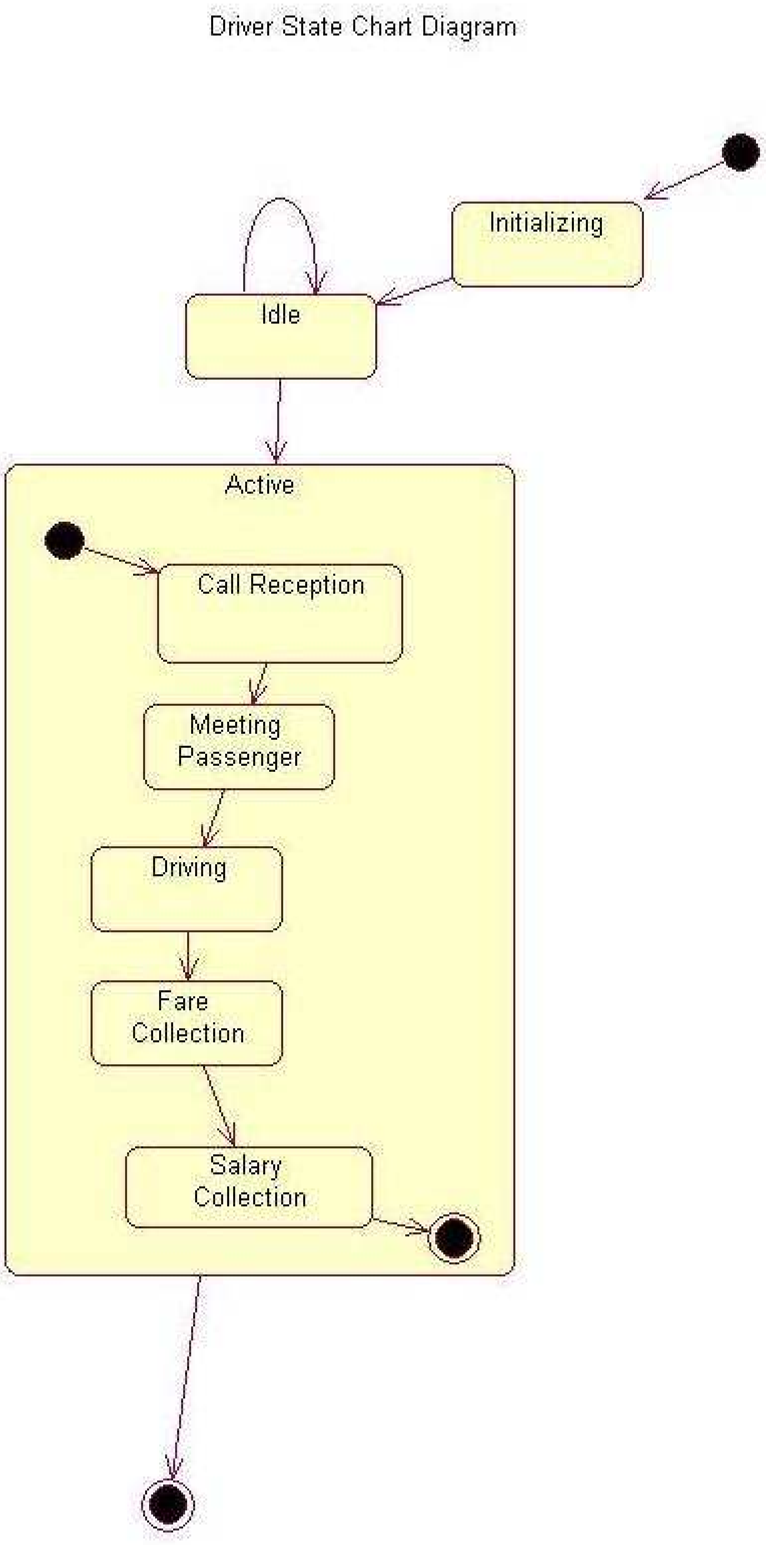

Figure 48: Driver State chart diagram

9..7 Component Diagram for Cab Management System

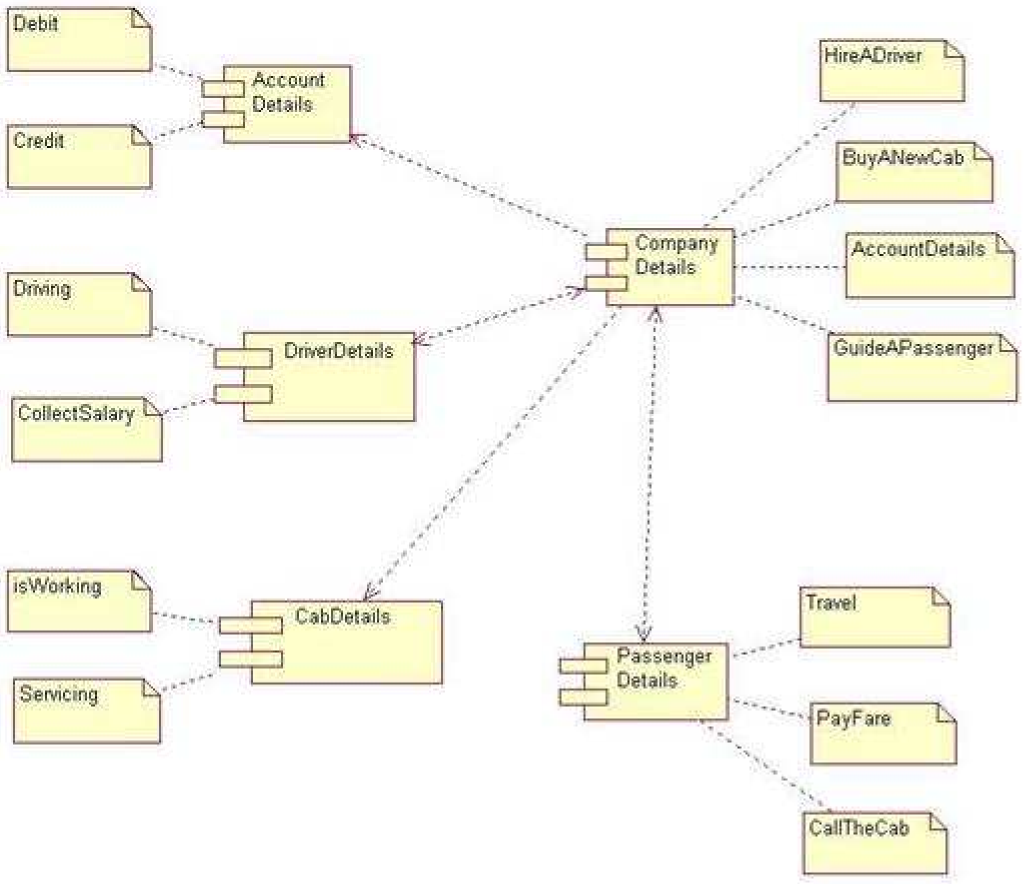

Figure 49: Component diagram for Cab Management System

9..8 Deployment Diagram

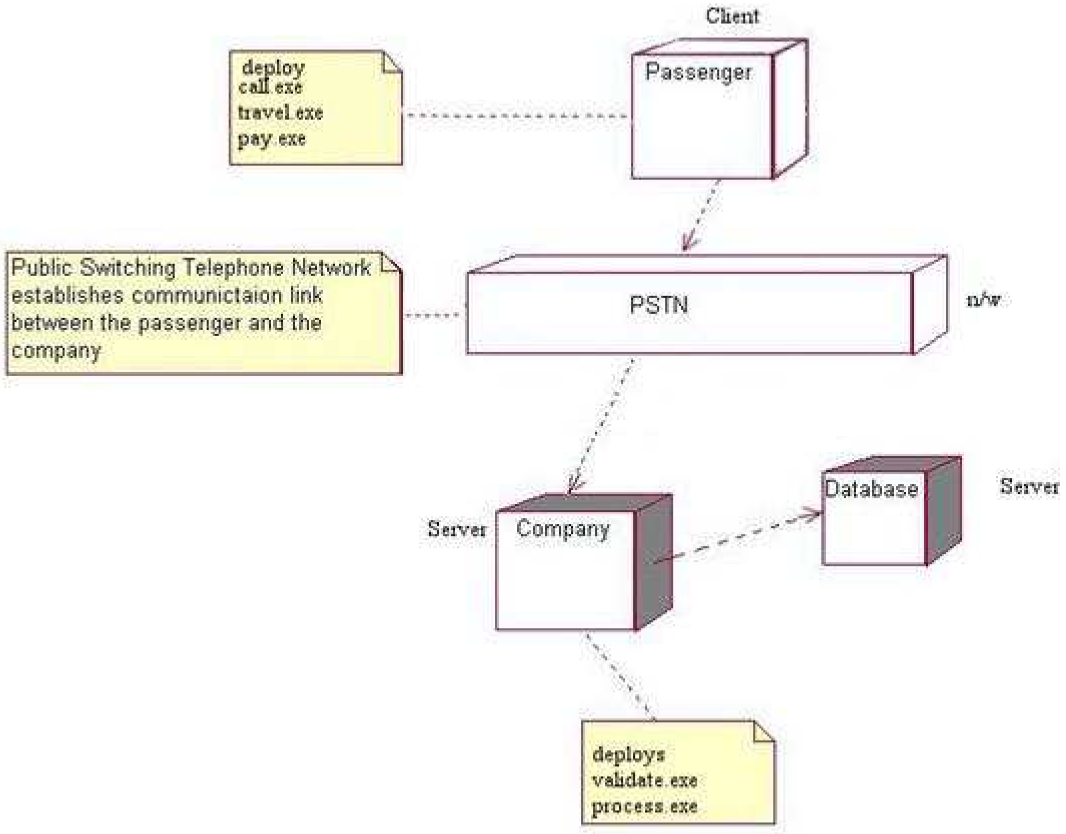

Figure 50: Deployment diagram for Cab Management System

10. Other Case Studies (Abstracts)

1. E – Seva: The case discussed E-seva (Electronic services), is an e-government project initiated by the government of Andhra Pradesh in India. The e-seva project aimed at integrated and offering a wide range of Government to citizen(G2C) services at single location. The case highlights the objectives of the E-seva project. The case also describes the G2C benefits of the citizen.

   Benefits:

   The implementation of the e-seva project benefits both the government as well as the citizen. The project enabled the government to provide services quickly and at low cost. Since the project introduced a new channel for interaction with the citizen the volume of G2C sevices increased. The services offered by ESEVA includes the payment of utility bills, registration of birth & death, issue of permits and licence, declaration of caste, income, recidence etc.,.

   Conclusion:

   a. ESEVA ....a true convergence of all the NeGP initiatives in rendering G2C services in a fast and secure way.
   b. Revamping of many moribund process and approaches.
   c. A precursor to the Right to Services Act.
   d. A model like ESEVA for the country offers the most advanced art, simple and cost-effective solution to meet the aspiration of the teeming millions in the country.

2. Outpatient Management in a Hospital: This case study includes the Outpatient details in Hospital management includes registrations of patients, storing their details into the system.Our software has a facility to give a unique id for every patient and store the details of every patient and staff automatically.User can search availability of doctor and the details of a patient using it.

   Benefits:

   The Hospital Management system can be entered using a username and password. It is accessible either by an administrator or receptions .The data can be retrieved easily. The interface is user friendly for both doctors and patients.

   Conclusion:

   This case study provides the technical information about doctors and patients in a hospital.In this case study a user/patient gets total information about doctor's availability in hospital and the detail information about hospital, specialist in hospital.It also give information about services and facilities available in the hospital.

Annexure – I

List of programs according to syllabus(Osmania University).

PART 1

Introduction to UML
Introduction to Rational Rose

1. Submission of Abstract of the Case Study
2. Usecase Diagram
3. Activity Diagram
4. Sequence Diagram
5. Collaboration Diagram
6. State Transition Diagram
7. Class Diagram
8. Component Diagram
9. Deployment Diagram

PART 2

Documentation

10. Data Flow Diagram
11. Dynamic Modeling using Finite State Automata
12. Software Requirement Specification
13. Functional Decomposition and Structure
14. Data Dictionary,Verification, Module Specification,CRE, User Manual