Importance of Modeling:

- If you want to **build a dog house**, you can pretty much start with a pile of lumber, some nails, and a few basic tools, such as a hammer, saw, and tape measure.
- In a few hours, with little prior planning, you'll likely end up with a dog house that's reasonably functional, and you can probably do it with no one else's help.
- As long as it's big enough and doesn't leak too much, your dog will be happy. If it doesn't work out, you can always start over, or get a less demanding dog.
- If you want to **build a house for your family**, you can start with a pile of lumber, some nails, and a few basic tools, but it's going to take you a lot longer, and your family will certainly be more demanding than the dog.
- In this case, unless you've already done it a few dozen times before, you'll be better served by doing some detailed planning before you pound the first nail or lay the foundation.
- At the very least, you'll want to make some sketches of how you want the house to look.
- If you want to build a quality house that meets the needs of your family and of local building codes, you'll need to draw some blueprints as well, so that you can think through the intended use of the rooms and the practical details of lighting, heating, and plumbing.
- Given these plans, you can start to make reasonable estimates of the amount of time and materials this job will require.
- Although it is humanly possible to build a house yourself, you'll find it is much more efficient to work with others, possibly subcontracting out many key work products or buying prebuilt materials.
- As long as you stay true to your plans and stay within the limitations of time and money, your family will most likely be satisfied. If it doesn't work out, you can't exactly get a new family, so it is best to set expectations early and manage change carefully.
- If you want to **build a high-rise office building**, it would be infinitely stupid for you to start with a pile of lumber, some nails, and a few basic tools.

- Because you are probably using other people's money, they will demand to have input into the size, shape, and style of the building. Often, they will change their minds, even after you've started building.
- You will want to do extensive planning, because the cost of failure is high.
- You will be just a part of a much larger group responsible for developing and deploying the building, and so the team will need all sorts of blueprints and models to communicate with one another.
- As long as you get the right people and the right tools and actively manage the process of transforming an architectural concept into reality, you will likely end up with a building that will satisfy its tenants.
- If you want to keep building buildings, then you will want to be certain to balance the desires of your tenants with the realities of building technology, and you will want to treat the rest of your team professionally, never placing them at any risk or driving them so hard that they burn out.

What, then, is a model? Simply put,

- *A model is a simplification of reality.*
  - A model provides the blueprints of a system. Models may encompass detailed plans, as well as more general plans that give a 30,000-foot view of the system under consideration.
  - A good model includes those elements that have broad effect and omits those minor elements that are not relevant to the given level of abstraction. Every system may be described from different aspects using different models, and each model is therefore a semantically closed abstraction of the system.
  - A model may be structural, emphasizing the organization of the system, or it may be behavioral, emphasizing the dynamics of the system.

Why do we model?

- *We build models so that we can better understand the system we are developing.*
- Through modeling, we achieve four aims.
  1. Models help us to visualize a system as it is or as we want it to be.
  2. Models permit us to specify the structure or behavior of a system.

3. Models give us a template that guides us in constructing a system.
4. Models document the decisions we have made.

- Modeling is not just for big systems. Even the software equivalent of a dog house can benefit from some modeling. However, it's definitely true that the larger and more complex the system, the more important modeling becomes, for one very simple reason:
- *We build models of complex systems because we cannot comprehend such a system in its entirety.*
    - There are limits to the human ability to understand complexity. Through modeling, we narrow the problem we are studying by focusing on only one aspect at a time.
    - This is essentially the approach of "divide-and-conquer" that Edsger Dijkstra spoke of years ago: Attack a hard problem by dividing it into a series of smaller problems that you can solve.
    - Furthermore, through modeling, we amplify the human intellect. A model properly chosen can enable the modeler to work at higher levels of abstraction.

Principles of Modeling:

- The use of modeling has a rich history in all the engineering disciplines. That experience suggests four basic principles of modeling.
- **First**, *The choice of what models to create has a profound influence on how a problem is attacked and how a solution is shaped.*
    - In other words, choose your models well. The right models will brilliantly illuminate the most wicked development problems, offering insight that you simply could not gain otherwise; the wrong models will mislead you, causing you to focus on irrelevant issues.
- **Second**,*Every model may be expressed at different levels of precision.*
    - If you are building a high rise, sometimes you need a 30,000-foot view for instance, to help your investors visualize its look and feel. Other times, you

need to get down to the level of the studs for instance, when there's a tricky pipe run or an unusual structural element.

- **Third**, *The best models are connected to reality.*
  - A physical model of a building that doesn't respond in the same way as do real materials has only limited value; a mathematical model of an aircraft that assumes only ideal conditions and perfect manufacturing can mask some potentially fatal characteristics of the real aircraft. It's best to have models that have a clear connection to reality, and where that connection is weak, to know exactly how those models are divorced from the real world. All models simplify reality; the trick is to be sure that your simplifications don't mask any important details.
- **Fourth**, *No single model is sufficient. Every nontrivial system is best approached through a small set of nearly independent models.*
  - If you are constructing a building, there is no single set of blueprints that reveal all its details. At the very least, you'll need floor plans, elevations, electrical plans, heating plans, and plumbing plans.
  -

Object Oriented Modeling:

- In software, there are several ways to approach a model. The two most common ways are from an algorithmic perspective and from an object-oriented perspective.
- The traditional view of software development takes an **algorithmic perspective**. In this approach, the main building block of all software is the procedure or function.
- This view leads developers to focus on issues of control and the decomposition of larger algorithms into smaller ones. There's nothing inherently evil about such a point of view except that it tends to yield brittle systems.
- As requirements change (and they will) and the system grows (and it will), systems built with an algorithmic focus turn out to be very hard to maintain.
- The contemporary view of software development takes an **object-oriented perspective**. In this approach, the main building block of all software systems is the object or class.
- Simply put, an object is a thing, generally drawn from the vocabulary of the problem space or the solution space; a class is a description of a set of common objects.

- Every object has identity (you can name it or otherwise distinguish it from other objects), state (there's generally some data associated with it), and behavior (you can do things to the object, and it can do things to other objects, as well).
- For example, consider a **simple three-tier architecture** for a billing system, involving a user interface, middleware, and a database.
- In the **user interface**, you will find concrete objects, such as buttons, menus, and dialog boxes.
- In the **database**, you will find concrete objects, such as tables representing entities from the problem domain, including customers, products, and orders.
- In the **middle layer**, you will find objects such as transactions and business rules, as well as higher-level views of problem entities, such as customers, products, and orders.

Building Blocks of UML:

- The vocabulary of the UML encompasses three kinds of building blocks:
    1. Things
    2. Relationships
    3. Diagrams
- Things are the abstractions that are first-class citizens in a model; relationships tie these things together; diagrams group of interesting collections of things.

I.Things in the UML

- There are four kinds of things in the UML:
    1. Structural things
    2. Behavioral things
    3. Grouping things
    4. Annotational things
- These things are the basic object-oriented building blocks of the UML. You use them to write well formed models.

1. Structural Things

- *Structural things* are the nouns of UML models. These are the mostly static parts of a model, representing elements that are either conceptual or physical. In all, there are seven kinds of structural things.

i. Class:

- It is a description of a set of objects that share the same attributes, operations, relationships, and semantics. A class implements one or more interfaces.
- Graphically, a class is rendered as a rectangle, usually including its name, attributes, and operations, as in Figure 2-1.

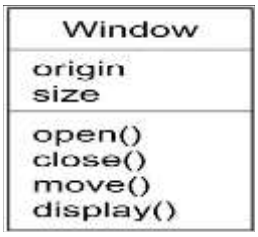


Figure 2-1 Classes

ii. Interface:

- *It* is a collection of operations that specify a service of a class or component.
- An interface defines a set of operation specifications (that is, their signatures) but never a set of operation implementations.
- Graphically, an interface is rendered as a circle together with its name. An interface rarely stands alone. Rather, it is typically attached to the class or component that realizes the interface, as in Figure 2-2.

Figure 2-2 Interfaces

iii. Collaboration:

- It defines an interaction and is a society of roles and other elements that work together to provide some cooperative behavior that's bigger than the sum of all the elements.
- Graphically, a collaboration is rendered as an ellipse with dashed lines, usually including only its name, as in Figure 2-3.

Figure 2-3 Collaborations

iv. use case:

- *It* is a description of sequences of actions that a system performs that yields an observable result of value to a particular actor.
- Graphically, a use case is rendered as an ellipse with solid lines, usually including only its name, as in Figure 2-4.

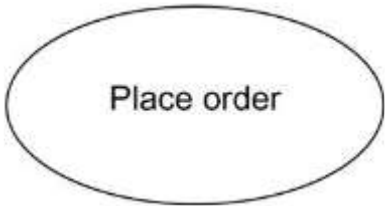


Figure 2-4 Use Cases

v. Active class:

- *It* is a class whose objects own one or more processes or threads and therefore can initiate control activity. An active class is just like a class except that its objects represent elements whose behavior is concurrent with other elements.
- Graphically, an active class is rendered just like a class, but with heavy lines, usually including its name, attributes, and operations, as in Figure 2-5.

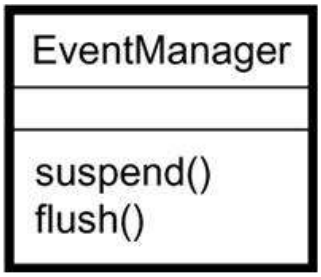


Figure 2-5 Active Classes

vi. Component:

- *It* is a physical and replaceable part of a system that conforms to and provides the realization of a set of interfaces. In a system,
- Graphically, a component is rendered as a rectangle with tabs, usually including only its name, as in Figure 2-6.

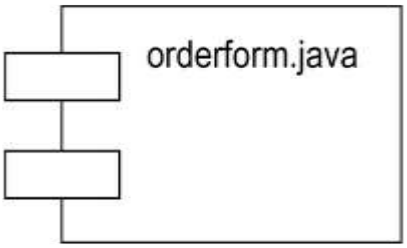


Figure 2-6 Components

vii. Node:

- ❖ *It* is a physical element that exists at run time and represents a computational resource, generally having at least some memory and, often, processing capability.
- ❖ Graphically, a node is rendered as a cube, usually including only its name, as in Figure 2-8.

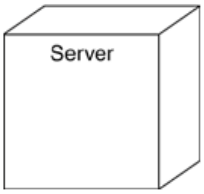


Figure 2-8. Nodes

2. Behavioral Things

- ➢ Behavioral things are the dynamic parts of UML models. These are the verbs of a model, representing behavior over time and space. In all, there are three primary kinds of behavioral things.

i. *Interaction:*

- ❖ It is a behavior that comprises a set of messages exchanged among a set of objects or roles within a particular context to accomplish a specific purpose.
- ❖ An interaction involves a number of other elements, including messages, actions, and connectors (the connection between objects).
- ❖ Graphically, a message is rendered as a directed line, almost always including the name of its operation, as in Figure 2-9.

Figure 2-9. Messages

ii. ***state machine:***

- ❖ It is a behavior that specifies the sequences of states an object or an interaction goes through during its lifetime in response to events, together with its responses to those events.
- ❖ The behavior of an individual class or a collaboration of classes may be specified with a state machine.
- ❖ A state machine involves a number of other elements, including states, transitions (the flow from state to state), events (things that trigger a transition), and activities (the response to a transition).
- ❖ Graphically, a state is rendered as a rounded rectangle, usually including its name and its substates, if any, as in Figure 2-10.

Figure 2-10. States

3. Grouping Things

- ➢ Grouping things are the organizational parts of UML models. These are the boxes into which a model can be decomposed. There is one primary kind of grouping thing, namely, packages.

i. Package:

- ❖ It is a general-purpose mechanism for organizing the design itself, as opposed to classes, which organize implementation constructs.
- ❖ Structural things, behavioral things, and even other grouping things may be placed in a package.
- ❖ Graphically, a package is rendered as a tabbed folder, usually including only its name and, sometimes, its contents, as in Figure 2-12.

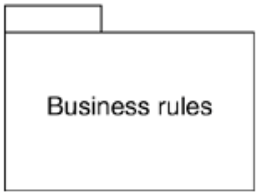


Figure 2-12. Packages

4. Annotational Things

- Annotational things are the explanatory parts of UML models. These are the comments one may apply to describe, illuminate, and remark about any element in a model.

i. Note:

- *It* is simply a symbol for rendering constraints and comments attached to an element or a collection of elements.
- Graphically, a note is rendered as a rectangle with a dog-eared corner, together with a textual or graphical comment, as in Figure 2-13.

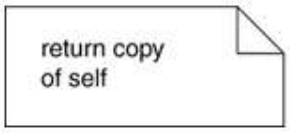


Figure 2-13. Notes

II. Relationships in the UML

- There are four kinds of relationships in the UML:

1. Dependency
2. Association
3. Generalization
4. Realization

- These relationships are the basic relational building blocks of the UML. One use them to write well-formed models.

1. Dependency:

- It is a semantic relationship between two model elements in which a change to one element (the independent one) may affect the semantics of the other element (the dependent one).
- Graphically, a dependency is rendered as a dashed line, possibly directed, and occasionally including a label, as in Figure 2-14.

Figure 2-14. Dependencies

2. Association:

- It is a structural relationship among classes that describes a set of links, a link being a connection among objects that are instances of the classes.
- Aggregation is a special kind of association, representing a structural relationship between a whole and its parts.
- Graphically, an association is rendered as a solid line, possibly directed, occasionally including a label, and often containing other adornments, such as multiplicity and end names, as in Figure 2-15.

Figure 2-15. Associations

3. Generalization:

- It is a specialization/generalization relationship in which the specialized element (the child) builds on the specification of the generalized element (the parent).
- The child shares the structure and the behavior of the parent.
- Graphically, a generalization relationship is rendered as a solid line with a hollow arrowhead pointing to the parent, as in Figure 2-16.

Figure 2-16. Generalizations

4. Realization*:*

- ❖ It is a semantic relationship between classifiers, wherein one classifier specifies a contract that another classifier guarantees to carry out.
- ❖ One'll encounter realization relationships in two places: between interfaces and the classes or components that realize them, and between use cases and the collaborations that realize them.
- ❖ Graphically, a realization relationship is rendered as a cross between a generalization and a dependency relationship, as in Figure 2-17.

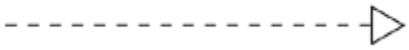


Figure 2-17. Realizations

III. Diagrams in the UML

- ➢ A ***diagram*** is the graphical presentation of a set of elements, most often rendered as a connected graph of vertices (things) and paths (relationships).
- ➢ The UML includes thirteen kinds of diagrams:

1. Class diagram
2. Object diagram
3. Use case diagram
4. Sequence diagram
5. Communication diagram
6. State diagram
7. Activity diagram
8. Component diagram
9. Deployment diagram

1. ***class diagram:***

   - ❖ It shows a set of classes, interfaces, and collaborations and their relationships.
   - ❖ These diagrams are the most common diagram found in modeling object-oriented systems.
   - ❖ Class diagrams address the static design view of a system.

2. ***object diagram:***

   - ❖ It shows a set of objects and their relationships.
   - ❖ Object diagrams represent static snapshots of instances of the things found in class diagrams.
   - ❖ These diagrams address the static design view or static process view of a system as do class diagrams, but from the perspective of real or prototypical cases.

3. ***use case diagram:***

   - ❖ It shows a set of use cases and actors (a special kind of class) and their relationships.
   - ❖ Use case diagrams address the static use case view of a system.
   - ❖ These diagrams are especially important in organizing and modeling the behaviors of a system.

4. ***sequence diagram:***

   - ❖ It is an interaction diagram that emphasizes the time-ordering of messages;

5. ***communication diagram:***

   - ❖ It is an interaction diagram that emphasizes the structural organization of the objects or roles that send and receive messages.

- ➢ Sequence diagrams and communication diagrams represent similar basic concepts, but each diagram emphasizes a different view of the concepts.

- Sequence diagrams emphasize temporal ordering, and communication diagrams emphasize the data structure through which messages flow.
- Both sequence diagrams and communication diagrams are kinds of interaction diagrams.
- An ***interaction diagram*** shows an interaction, consisting of a set of objects or roles, including the messages that may be dispatched among them. Interaction diagrams address the dynamic view of a system.

6. state diagram:

- It shows a state machine, consisting of states, transitions, events, and activities.
- A state diagrams shows the dynamic view of an object.
- They are especially important in modeling the behavior of an interface, class, or collaboration and emphasize the event-ordered behavior of an object, which is especially useful in modeling reactive systems

7. activity diagram:

- It shows the structure of a process or other computation as the flow of control and data from step to step within the computation.
- Activity diagrams address the dynamic view of a system. They are especially important in modeling the function of a system and emphasize the flow of control among objects.

8. component diagram:

- It shows an encapsulated class and its interfaces, ports, and internal structure consisting of nested components and connectors.
- Component diagrams address the static design implementation view of a system.
- They are important for building large systems from smaller parts. (UML distinguishes a composite structure diagram, applicable to any class, from a

component diagram, but we combine the discussion because the distinction between a component and a structured class is unnecessarily subtle.)

9. deployment diagram:

- It shows the configuration of run-time processing nodes and the components that live on them.
- Deployment diagrams address the static deployment view of an architecture. A node typically hosts one or more artifacts.

DIAGRAMS:

Terms and Concepts

- In modeling real systems, no matter what the problem domain, one'll find onesself creating the same kinds of diagrams, because they represent common views into common models.
- Typically, one'll view the static parts of a system using one of the following diagrams.

1. Class diagram
2. Object diagram
3. Component diagram
4. Deployment diagram

- One'll often use five additional diagrams to view the dynamic parts of a system.

1. Use case diagram
2. Sequence diagram
3. Collaboration diagram
4. State diagram
5. Activity diagram

Structural Diagrams

- The UML's structural diagrams exist to visualize, specify, construct, and document the static aspects of a system.
- One can think of the static aspects of a system as representing its relatively stable skeleton and scaffolding. Just as the static aspects of a house encompass the existence and placement of such things as walls, doors, windows, pipes, wires, and vents, so too do the static aspects of a software system encompass the existence and placement of such things as classes, interfaces, collaborations, components, and nodes.
- The UML's structural diagrams are roughly organized around the major groups of things one'll find when modeling a system.

1.Class diagram	Classes, interfaces, and collaborations
2.Object diagram	Objects
3.Component diagram	Components
4.Deployment diagram	Nodes

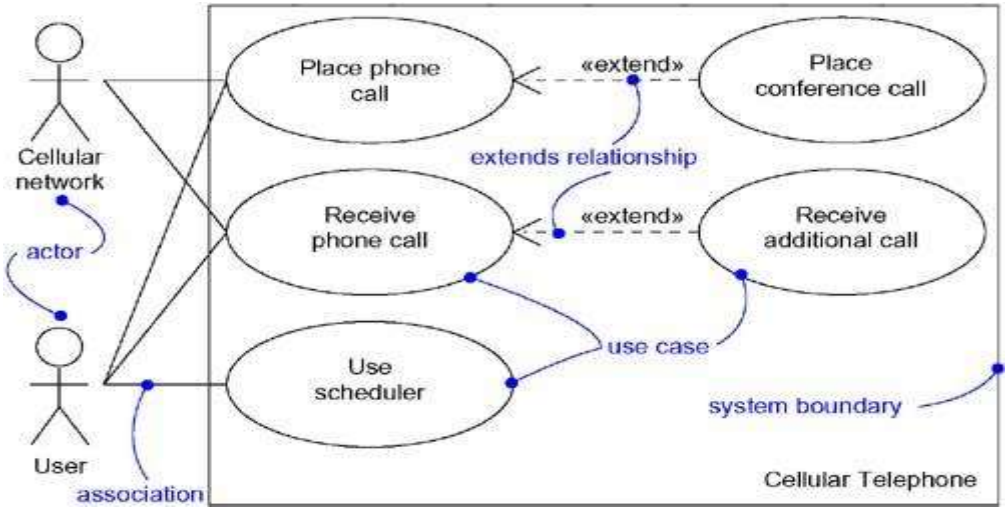
Basic Behavioural Modelling

Use Case Diagrams

Terms & Concepts

A use case diagram is a diagram that shows a set of use cases and actors and their relationships.

Figure 17-1 A Use Case Diagram



Common Properties

- A use case diagram is just a special kind of diagram and shares the same common properties as do all other diagrams - a name and graphical contents that are a projection into a model.

Contents

- Use case diagrams commonly contain
  - ✓ Use cases
  - ✓ Actors
  - ✓ Dependency, generalization, and association relationships
- Like all other diagrams, use case diagrams may contain notes and constraints.
- Use case diagrams may also contain packages, which are used to group elements of your model into larger chunks.

Common Uses

- You apply use case diagrams to model the static use case view of a system. This view primarily supports the behavior of a system• the outwardly visible services that the system provides in the context of its environment.
- When you model the static use case view of a system, you'll typically apply use case diagrams in one of two ways.

1. To model the context of a system

Modeling the context of a system involves drawing a line around the whole system and asserting which actors lie outside the system and interact with it.Here, you'll apply use case diagrams to specify the actors and the meaning of their roles.

2. To model the requirements of a system

Modeling the requirements of a system involves specifying what that system should do (from a point of view of outside the system), independent of how that system should do it.

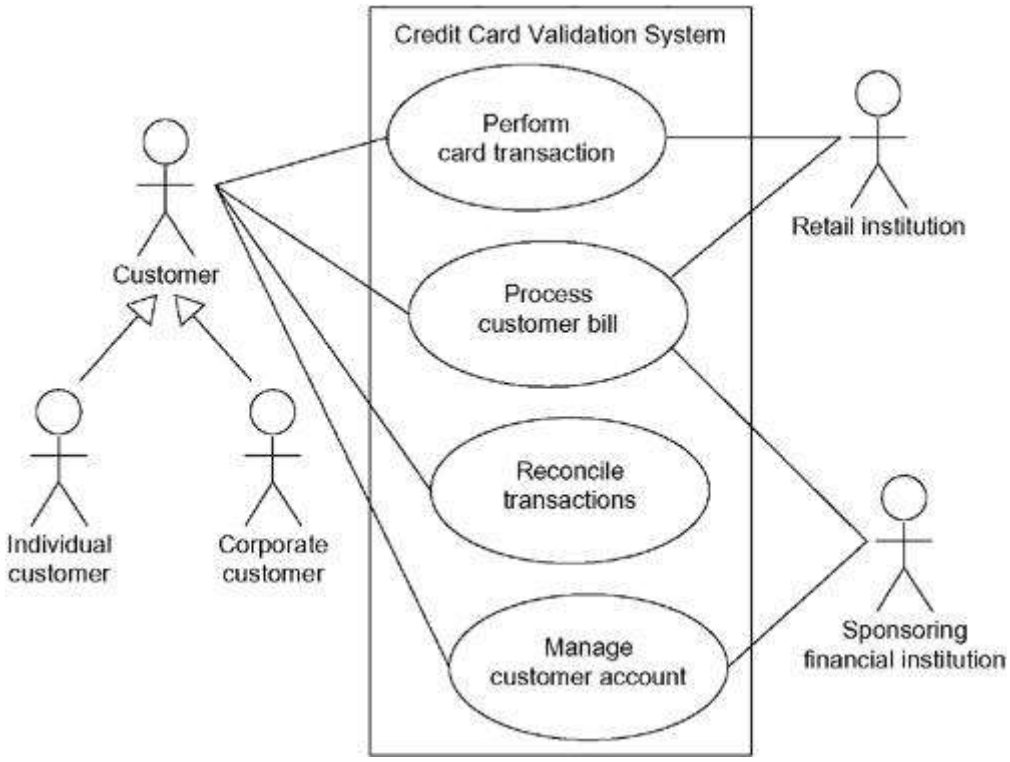
Common Modeling Techniques

Modeling the Context of a System

To model the context of a system,

- Identify the actors that surround the system by considering which groups require help from the system to perform their tasks; which groups are needed to execute the system's functions; which groups interact with external hardware or other software systems; and which groups perform secondary functions for administration and maintenance.
- Organize actors that are similar to one another in a generalization/specialization hierarchy.
- Where it aids understandability, provide a stereotype for each such actor.
- Populate a use case diagram with these actors and specify the paths of communication from each actor to the system's use cases.

Figure 17-2 Modeling the Context of a System

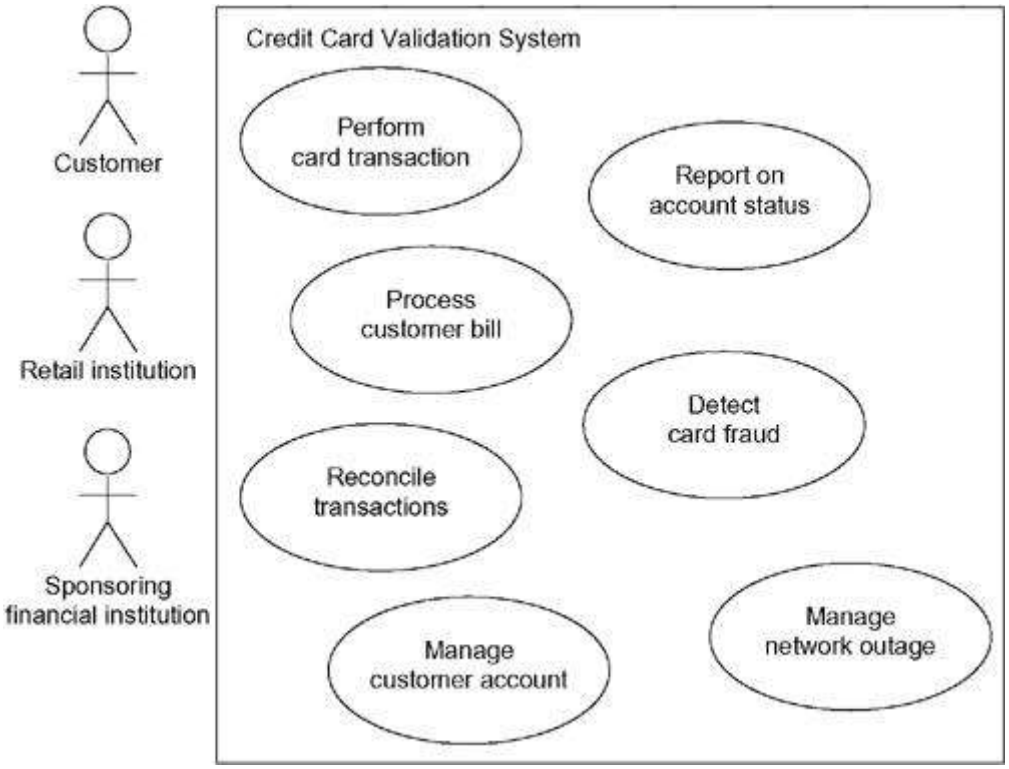


Modeling the Requirements of a System

To model the requirements of a system,

- Establish the context of the system by identifying the actors that surround it.
- For each actor, consider the behavior that each expects or requires the system to provide.
- Name these common behaviors as use cases.
- Factor common behavior into new use cases that are used by others; factor variant behavior into new use cases that extend more main line flows.
- Model these use cases, actors, and their relationships in a use case diagram.
- Adorn these use cases with notes that assert nonfunctional requirements; you may have to attach some of these to the whole system.

Figure 17-3 Modeling the Requirements of a System



Interaction Diagrams

Terms & Concepts

- An *interaction diagram* shows an interaction, consisting of a set of objects and their relationships, including the messages that may be dispatched among them.
- A *sequence diagram* is an interaction diagram that emphasizes the time ordering of messages. Graphically, a sequence diagram is a table that shows objects arranged along the X axis and messages, ordered in increasing time, along the Y axis.
- A *collaboration diagram* is an interaction diagram that emphasizes the structural organization of the objects that send and receive messages. Graphically, a collaboration diagram is a collection of vertices and arcs.

Common Properties

- An interaction diagram is just a special kind of diagram and shares the same common properties as do all other diagrams - a name and graphical contents that are a projection into a model.

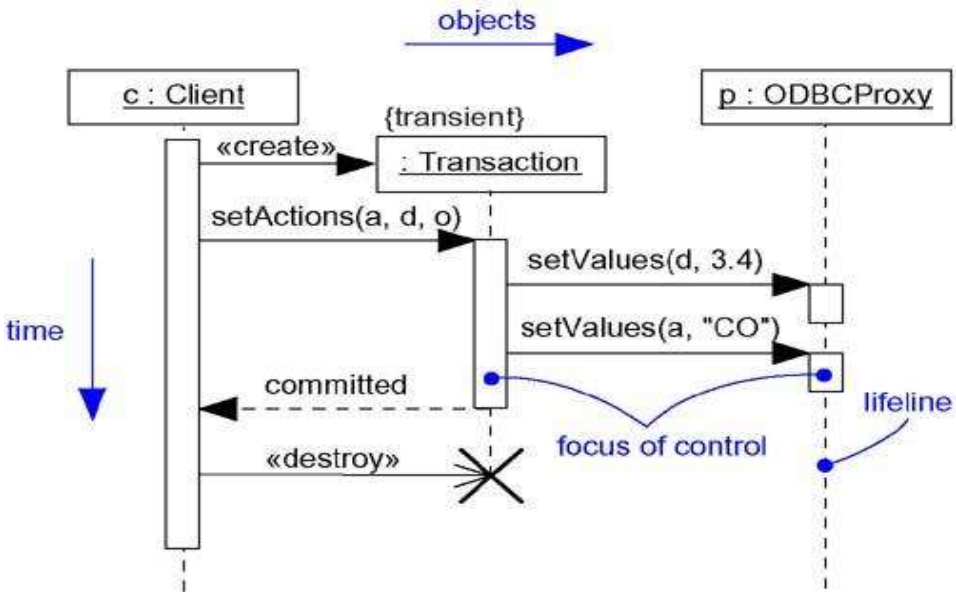
Contents

- Interaction diagrams commonly contain
  - ✓ Objects
  - ✓ Links
  - ✓ Messages
- Like all other diagrams, interaction diagrams may contain notes and constraints.

Sequence Diagrams

- A sequence diagram emphasizes the time ordering of messages.
- As Figure 18-2 shows, you form a sequence diagram by first placing the objects that participate in the interaction at the top of your diagram, across the X axis.
- Typically, you place the object that initiates the interaction at the left, and increasingly more subordinate objects to the right.
- Next, you place the messages that these objects send and receive along the Y axis, in order of increasing time from top to bottom. This gives the reader a clear visual cue to the flow of control over time.

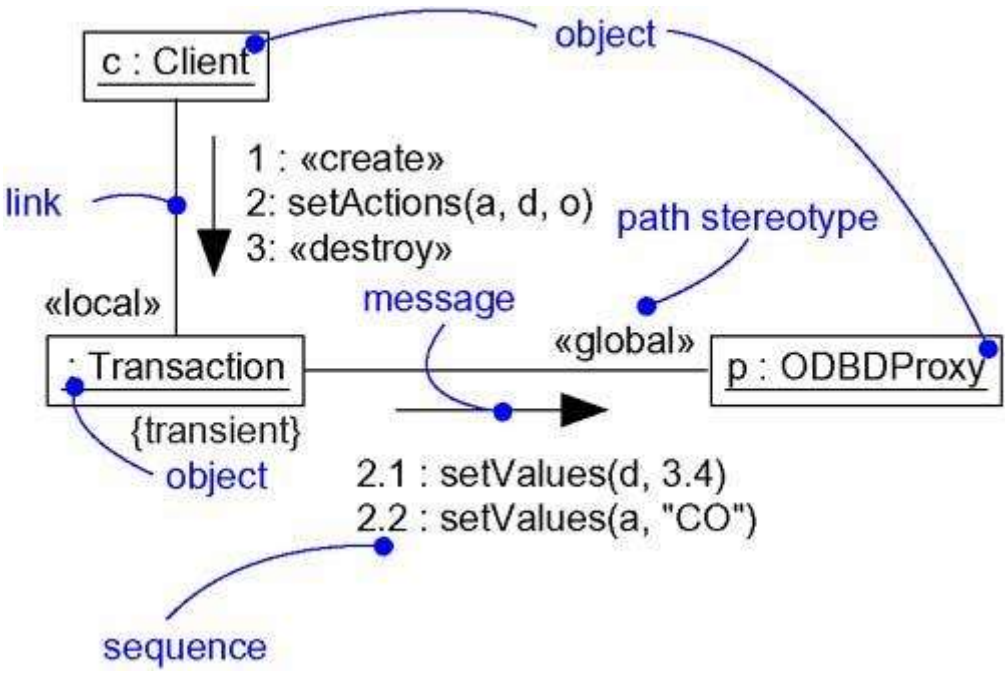
Figure 18-2 Sequence Diagram



Collaboration Diagrams

- A collaboration diagram emphasizes the organization of the objects that participate in an interaction.
- As Figure 18-3 shows, you form a collaboration diagram by first placing the objects that participate in the interaction as the vertices in a graph.
- Next, you render the links that connect these objects as the arcs of this graph.
- Finally, you adorn these links with the messages that objects send and receive. This gives the reader a clear visual cue to the flow of control in the context of the structural organization of objects that collaborate.

Figure 18-3 Collaboration Diagram



Semantic Equivalence

- Because they both derive from the same information in the UML's meta model, sequence diagrams and collaboration diagrams are semantically equivalent.
- As a result, you can take a diagram in one form and convert it to the other without any loss of information, as you can see in the previous two figures, which are semantically equivalent.

Common Uses

1. **To model flows of control by time ordering**
2. **To model flows of control by organization**

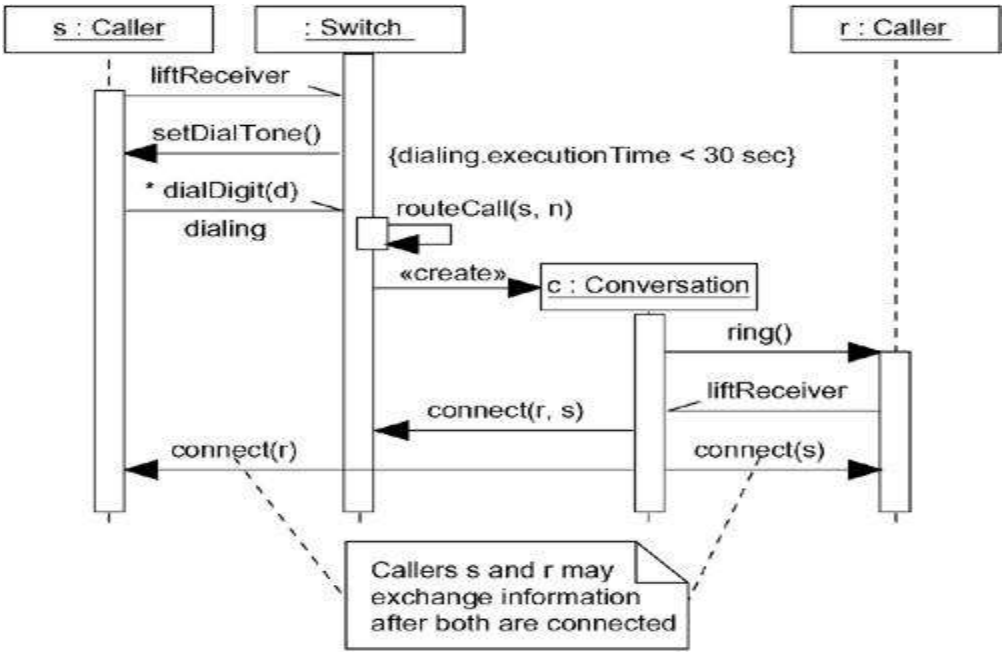
Common Modeling Techniques

Modeling Flows of Control by Time Ordering

To model a flow of control by time ordering,

- Set the context for the interaction, whether it is a system, subsystem, operation, or class, or one scenario of a use case or collaboration.
- Set the stage for the interaction by identifying which objects play a role in the interaction. Lay them out on the sequence diagram from left to right, placing the more important objects to the left and their neighboring objects to the right.
- Set the lifeline for each object. In most cases, objects will persist through the entire interaction. For those objects that are created and destroyed during the interaction, set their lifelines, as appropriate, and explicitly indicate their birth and death with appropriately stereotyped messages.
- Starting with the message that initiates this interaction, lay out each subsequent message from top to bottom between the lifelines, showing each message's properties (such as its parameters), as necessary to explain the semantics of the interaction.
- If you need to visualize the nesting of messages or the points in time when actual computation is taking place, adorn each object's lifeline with its focus of control.
- If you need to specify time or space constraints, adorn each message with a timing mark and attach suitable time or space constraints.
- If you need to specify this flow of control more formally, attach pre- and post conditions to each message.

Figure 18-4 Modeling Flows of Control by Time Ordering



Modeling Flows of Control by Organization

To model a flow of control by organization,

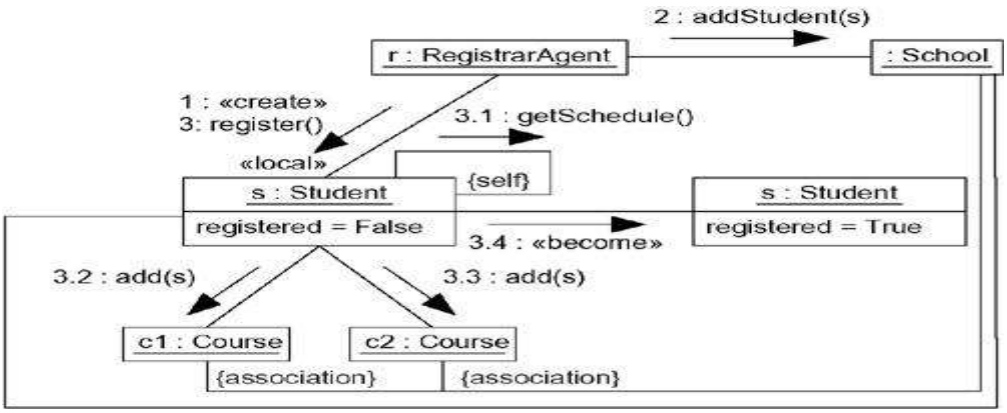
- Set the context for the interaction, whether it is a system, subsystem, operation, or class,

  or one scenario of a use case or collaboration.
- Set the stage for the interaction by identifying which objects play a role in the interaction. Lay them out on the collaboration diagram as vertices in a graph, placing the more important objects in the center of the diagram and their neighboring objects to the outside.
- Set the initial properties of each of these objects. If the attribute values, tagged values, state, or role of any object changes in significant ways over the duration of the interaction, place a duplicate object on the diagram, update it with these new values, and connect them by a message stereotyped as become or copy (with a suitable sequence number).
- Specify the links among these objects, along which messages may pass.
    1. Lay out the association links first; these are the most important ones, because they represent structural connections.

2. Lay out other links next, and adorn them with suitable path stereotypes (such as global and local) to explicitly specify how these objects are related to one another.

- Starting with the message that initiates this interaction, attach each subsequent message to the appropriate link, setting its sequence number, as appropriate. Show nesting by using Dewey decimal numbering.
- If you need to specify time or space constraints, adorn each message with a timing mark and attach suitable time or space constraints.
- If you need to specify this flow of control more formally, attach pre- and postconditions to each message.

Figure 18-5 Modeling Flows of Control by Organization

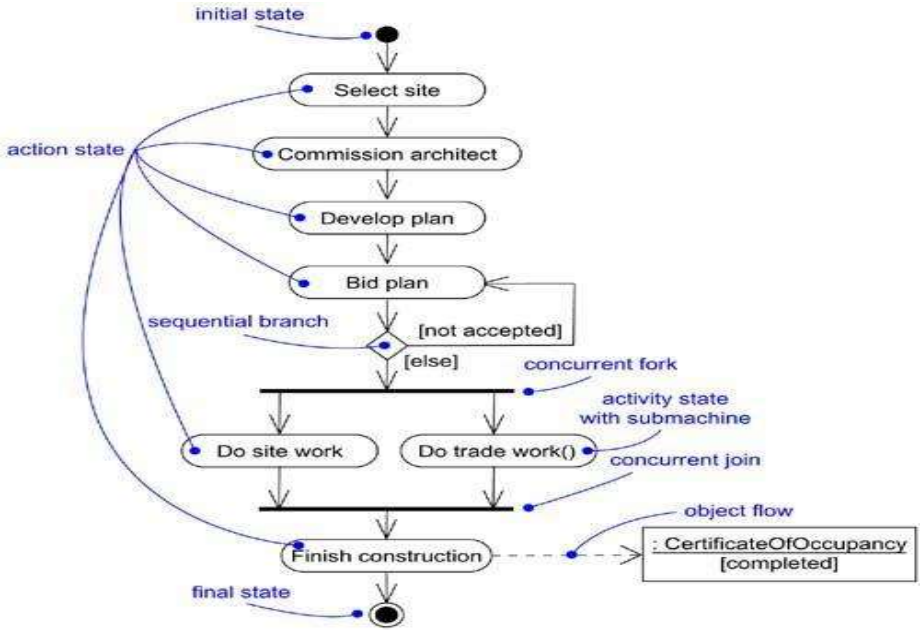


Activity Diagrams

Terms & Concepts

- An activity diagram shows the flow from activity to activity. An *activity diagram* is an ongoing nonatomic execution within a state machine.

Figure 19-1 Activity Diagrams



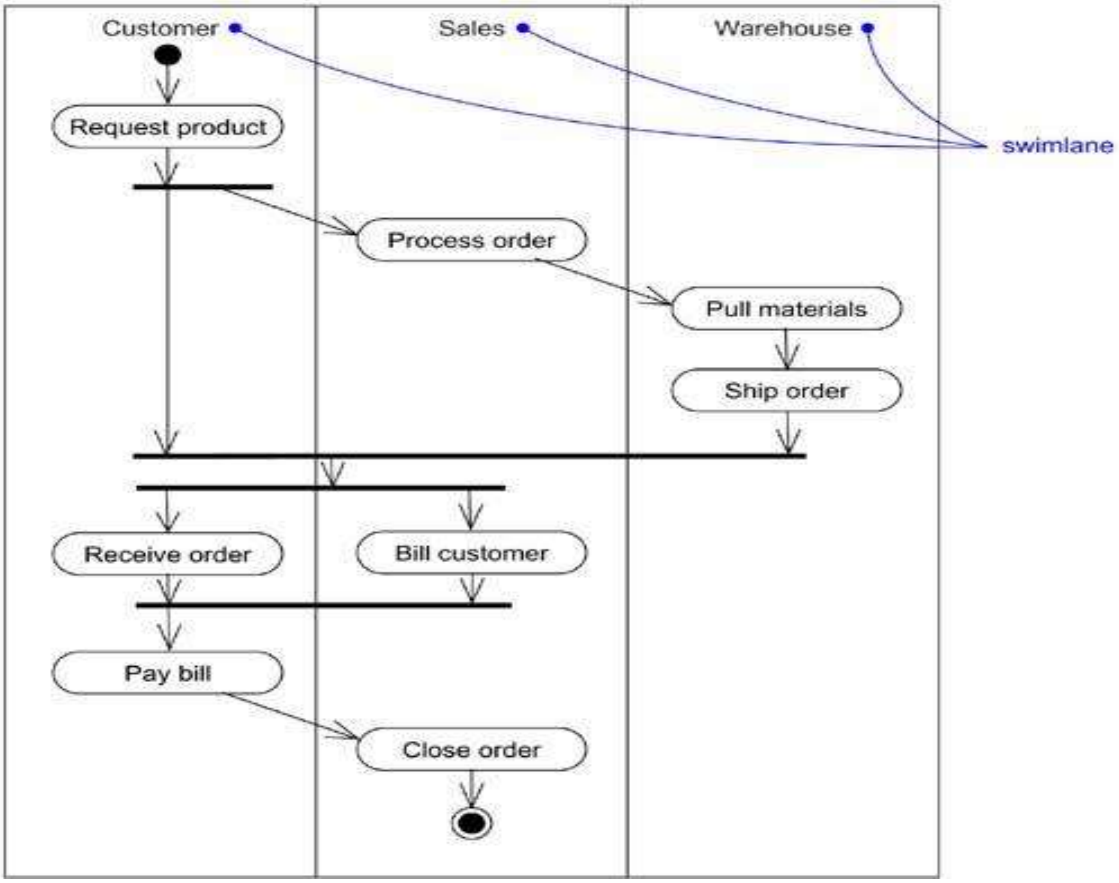
Common Properties

- An activity diagram is just a special kind of diagram and shares the same common properties as do all other diagrams - a name and graphical contents that are a projection into a model.

Contents

- Activity diagrams commonly contain
  - ✓ Activity states and action states
  - ✓ Transitions
  - ✓ Objects

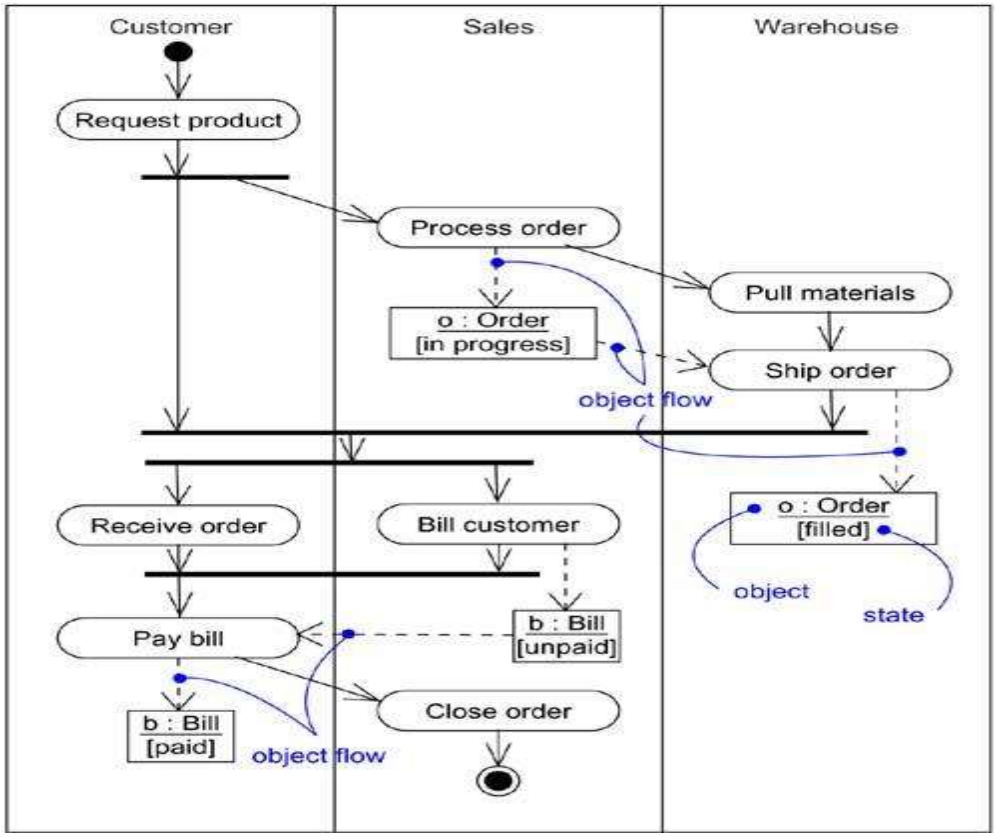
Figure 19-7 Swimlanes



Object Flow

- As Figure 19-8 shows, you can specify the things that are involved in an activity diagram by placing these objects in the diagram, connected using a dependency to the activity or transition that creates, destroys, or modifies them.
- This use of dependency relationships and objects is called an object flow because it represents the participation of an object in a flow of control.

Figure 19-8 Object Flow



Common Uses

- You use activity diagrams to model the dynamic aspects of a system.
- When you model the dynamic aspects of a system, you'll typically use activity diagrams in two ways.

  1. To model a workflow

  2. To model an operation

Common Modeling Techniques

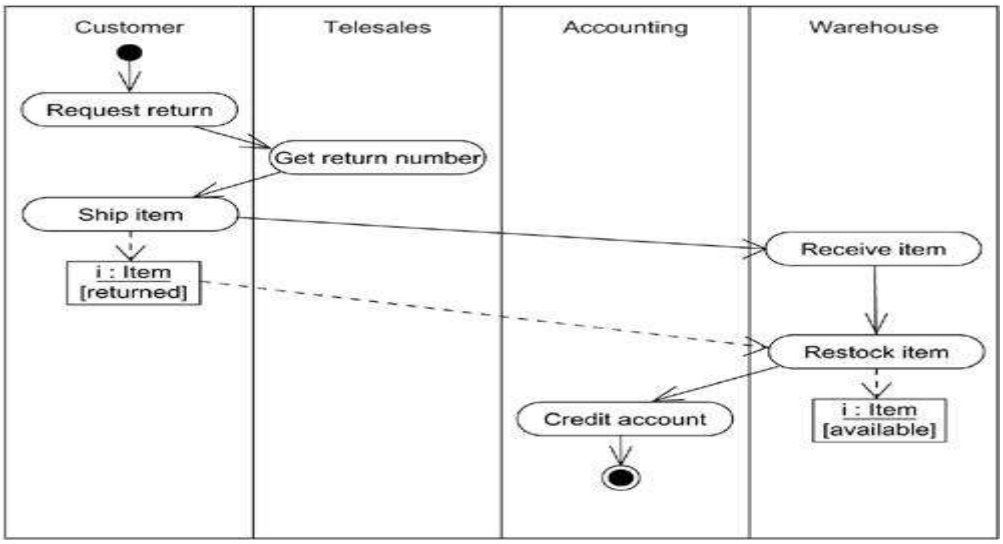
Modeling a Workflow

To model a workflow,

- Establish a focus for the workflow. For nontrivial systems, it's impossible to show all interesting workflows in one diagram.

- Select the business objects that have the high-level responsibilities for parts of the overall workflow. These may be real things from the vocabulary of the system, or they may be more abstract. In either case, create a swimlane for each important business object.
- Identify the preconditions of the workflow's initial state and the postconditions of the workflow's final state. This is important in helping you model the boundaries of the workflow.
- Beginning at the workflow's initial state, specify the activities and actions that take place over time and render them in the activity diagram as either activity states or action states.
- For complicated actions, or for sets of actions that appear multiple times, collapse these into activity states, and provide a separate activity diagram that expands on each.
- Render the transitions that connect these activity and action states. Start with the sequential flows in the workflow first, next consider branching, and only then consider forking and joining.
- If there are important objects that are involved in the workflow, render them in the activity diagram, as well. Show their changing values and state as necessary to communicate the intent of the object flow.

Figure 19-9 Modeling a Workflow

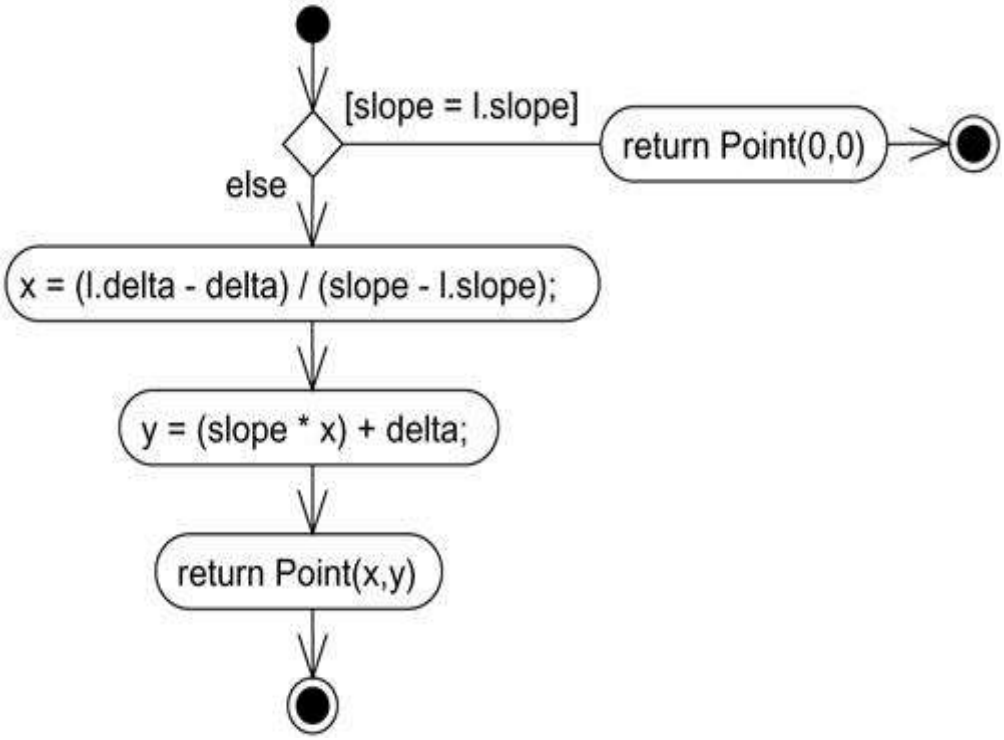


Modeling an Operation

To model an operation,

- Collect the abstractions that are involved in this operation. This includes the operation's parameters (including its return type, if any), the attributes of the enclosing class, and certain neighboring classes.
- Identify the preconditions at the operation's initial state and the postconditions at the operation's final state. Also identify any invariants of the enclosing class that must hold during the execution of the operation.
- Beginning at the operation's initial state, specify the activities and actions that take place over time and render them in the activity diagram as either activity states or action states.
- Use branching as necessary to specify conditional paths and iteration.
- Only if this operation is owned by an active class, use forking and joining as necessary to specify parallel flows of control.

Figure 19-10 Modeling an Operation



Class Diagram

Terms and Concepts

- A *class diagram* is a diagram that shows a set of classes, interfaces, and collaborations and their relationships. Graphically, a class diagram is a collection of vertices and arcs.

Common Properties

- A class diagram is just a special kind of diagram and shares the same common properties as do all other diagrams - a name and graphical content that are a projection into a model. What distinguishes a class diagram from all other kinds of diagrams is its particular content.

Contents

- Class diagrams commonly contain the following things:
  - ✓ Classes
  - ✓ Interfaces
  - ✓ Collaborations
  - ✓ Dependency, generalization, and association relationships
  - ✓ Like all other diagrams, class diagrams may contain notes and constraints.

Common Uses

- When you model the static design view of a system, you'll typically use class diagrams in one of three ways.
  1. **To model the vocabulary of a system**
  2. **To model simple collaborations**
  3. **To model a logical database schema**

Common Modeling Techniques

Modeling Simple Collaborations

- To model a collaboration,
  - Identify the mechanism you'd like to model. A mechanism represents some function or behavior of the part of the system you are modeling that results from the interaction of a society of classes, interfaces, and other things.
  - For each mechanism, identify the classes, interfaces, and other collaborations that participate in this collaboration. Identify the relationships among these things, as well.
  - Use scenarios to walk through these things. Along the way, you'll discover parts of your model that were missing and parts that were just plain semantically wrong.
  - Be sure to populate these elements with their contents. For classes, start with getting a good balance of responsibilities. Then, over time, turn these into concrete attributes and operations.

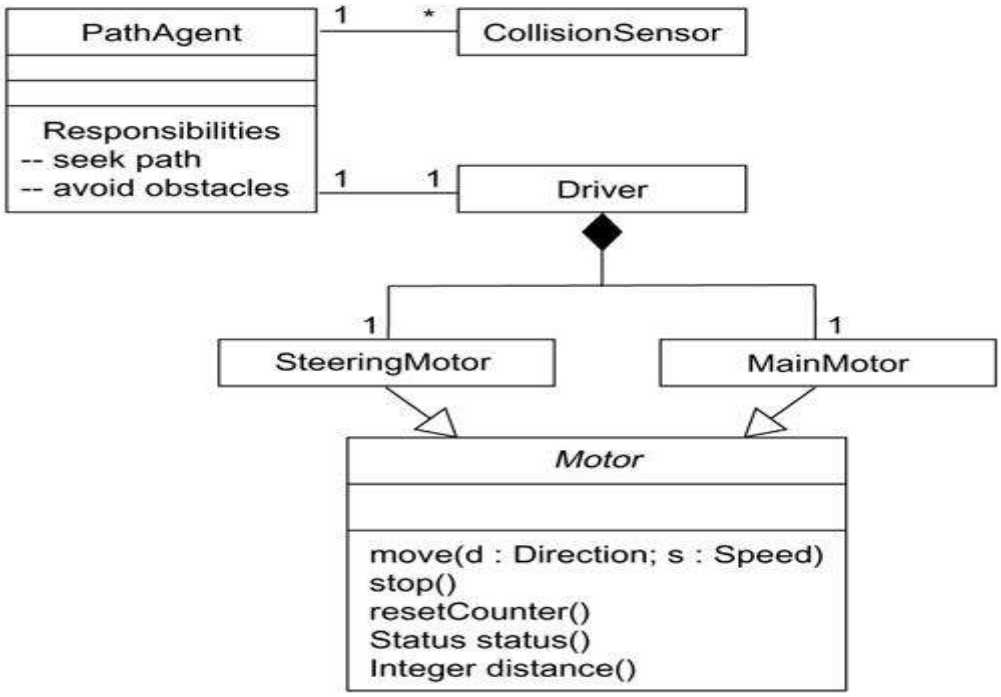


Figure 8-2 Modeling Simple Collaborations

Modeling a Logical Database Schema

- To model a schema,
  - Identify those classes in your model whose state must transcend the lifetime of their applications.
  - Create a class diagram that contains these classes and mark them as persistent (a standard tagged value). You can define your own set of tagged values to address database-specific details.
  - Expand the structural details of these classes. In general, this means specifying the details of their attributes and focusing on the associations and their cardinalities that structure these classes.
  - Watch for common patterns that complicate physical database design, such as cyclic associations, one-to-one associations, and n-ary associations. Where necessary, create intermediate abstractions to simplify your logical structure.
  - Consider also the behavior of these classes by expanding operations that are important for data access and data integrity. In general, to provide a better separation of concerns, business rules concerned with the manipulation of sets of these objects should be encapsulated in a layer above these persistent classes.
  - Where possible, use tools to help you transform your logical design into a physical design.

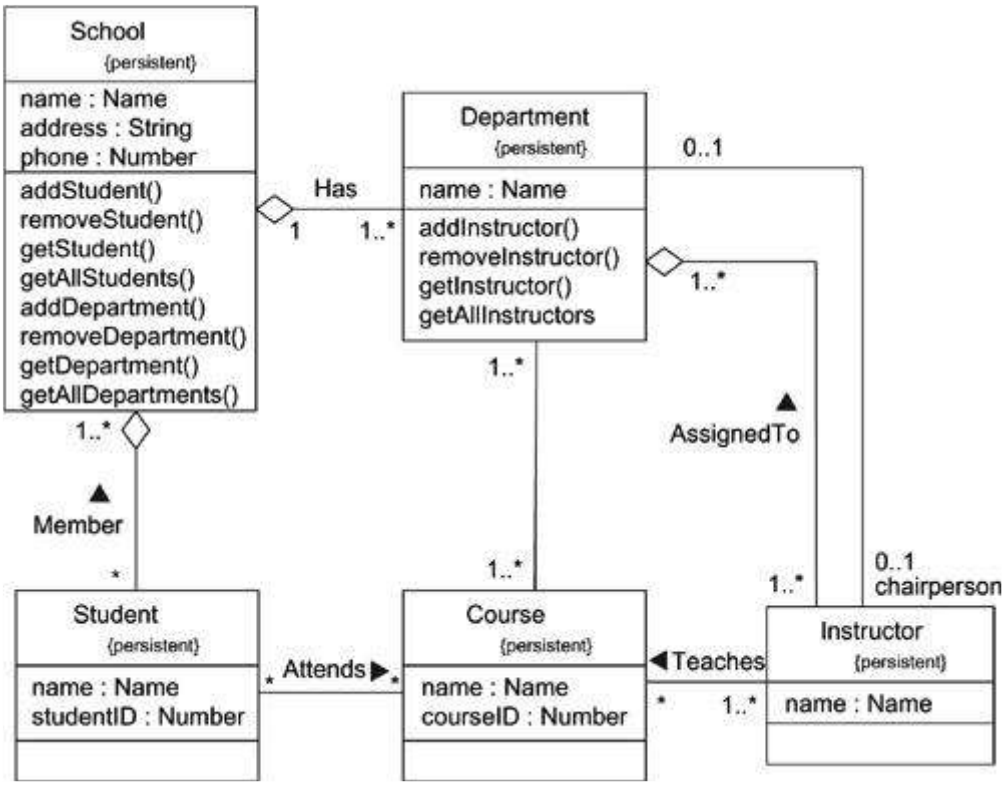
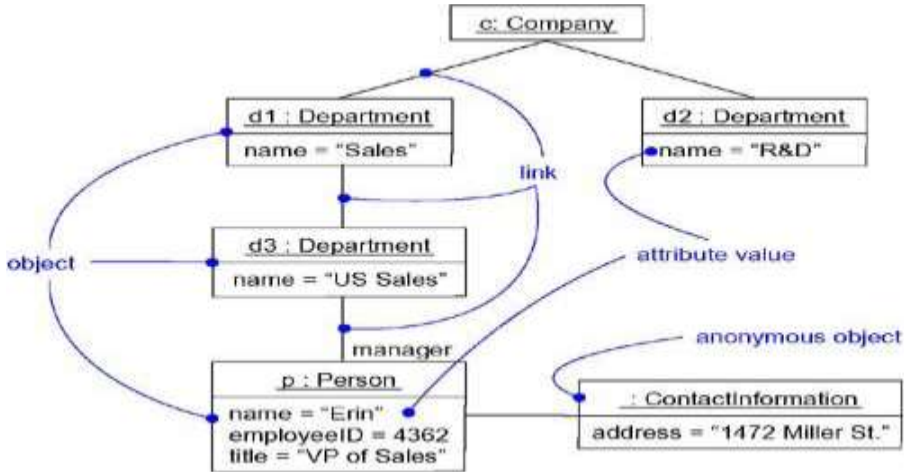


Figure 8-3 Modeling a Database Schema

Object Diagram

Terms and Concepts

- An *object diagram* is a diagram that shows a set of objects and their relationships at a point in time. Graphically, an object diagram is a collection of vertices and arcs



Common Properties

- An object diagram is a special kind of diagram and shares the same common properties as all other diagrams - that is, a name and graphical contents that are a projection into a model. What distinguishes an object diagram from all other kinds of diagrams is its particular content.

Contents

- Object diagrams commonly contain
  - ✓ Objects
  - ✓ Links
  - ✓ Like all other diagrams, object diagrams may contain notes and constraints.

Common Uses

- When you model the static design view or static process view of a system, you typically use object diagrams in one way:

  1. To model object structure

Common Modeling Techniques

Modeling Object Structures

- To model an object structure,
  - ❖ Identify the mechanism you'd like to model. A mechanism represents some function or behavior of the part of the system you are modeling that results from the interaction of a society of classes, interfaces, and other things.
  - ❖ For each mechanism, identify the classes, interfaces, and other elements that participate in this collaboration; identify the relationships among these things, as well.
  - ❖ Consider one scenario that walks through this mechanism. Freeze that scenario at a moment in time, and render each object that participates in the mechanism.

- Expose the state and attribute values of each such object, as necessary, to understand the scenario.
- Similarly, expose the links among these objects, representing instances of associations among them.

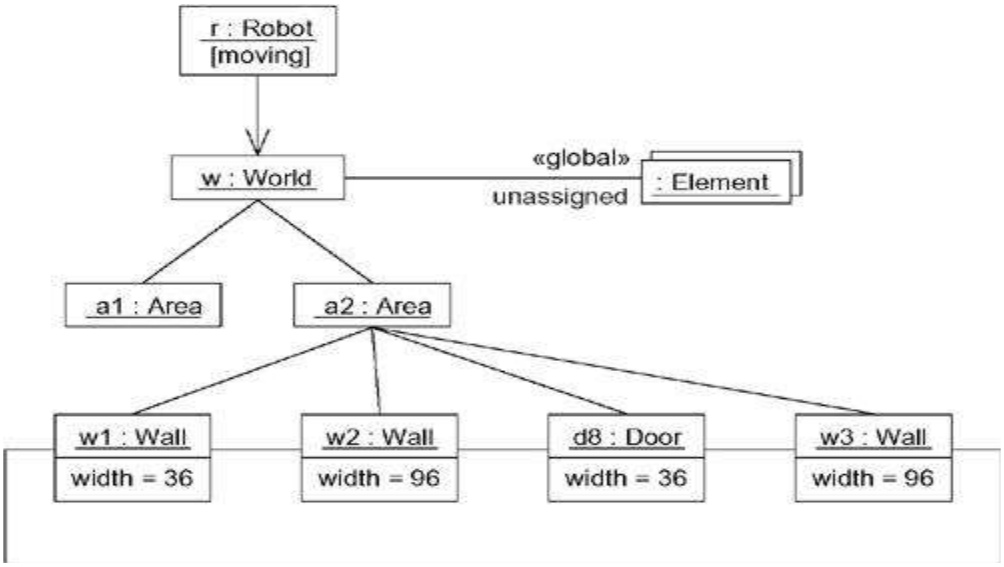


Figure 14-2 Modelling Object Structures

Project-I: A Point-of-Sale (POS) System is a computerized application used to record sales and handle payments; it is typically used in a retail store, it includes hardware components such as a computer and bar code scanner, and software to run the system. It interfaces to various service applications, such as a third-party tax calculator and inventory control. These systems must be relatively fault tolerant; that is, even if remote services are temporarily unavailable they must still be of capturing sales and handling at least cash payments. A POS system must support multiple and varied client-side terminals and interfaces such as browser, PDAs, touch-screens.

2. Point of Sale in detail: The **point of sale** (**POS**) or **point of purchase** (**POP**) is the time and place where a retail transaction is completed. At the point of sale, the merchant calculates the amount owed by the customer, indicates that amount, may prepare an invoice for the customer (which may be a cash register printout), and indicates the options for the customer to make payment. It is also the point at which a customer makes a payment to the merchant in exchange for goods or after provision of a service. After receiving payment, the merchant may issue a receipt for the transaction, which is usually printed but is increasingly being dispensed with or sent electronically.

To calculate the amount owed by a customer, the merchant may use various devices such as weighing scales, barcode scanners, and cash registers. To make a payment, payment terminals, touch screens, and other hardware and software options are available.

The point of sale is often referred to as the point of service because it is not just a point of sale but also a point of return or customer order. POS terminal software may also include features for additional functionality, such as inventory management, CRM, financials, or warehousing.

Businesses are increasingly adopting POS systems, and one of the most obvious and compelling reasons is that a POS system does away with the need for price tags. Selling prices are linked to the product code of an item when adding stock, so the cashier merely needs to scan this code to process a sale. If there is a price change, this can also be easily done through the inventory window. Other advantages include the ability to implement various types of discounts, a loyalty scheme for customers, and more efficient stock control.

3. Actors: There five actors who will use "point of sale system" . They are

- Manger
- Cashier
- Supplier
- Inventory System
- Customer

Manager: He generates sales reports and reorder depleted inventory.

Supplier: He is the one who receives the orders from manager and supplies the order on time.

Cashier: Who can check product availability, calculate total price and issue the product to customer.

Inventory System: Which can reorder depleted inventory, update inventory and check inventory.

Customer: The one who can request product, receives and do the payment.

4. Functional Requirements :

i. The system supports customers purchased receipt.
ii. System can search the product from the stock according to customers demand.
iii. System can add stock.
iv. System can update stock.
v. System can delete stock.
vi. System can show the stock report.
vii. System can show the sales report.
viii. System can register new staff.
ix. System can add customer service.
x. System can update customer service.
xi. System can view all the service records according to product specific ID.
xii. System can update password (Admin & Staff).

5. Non-Functional Requirements:

i. The system can save stock into the database safely (**data persistency**).
ii. The system can support all the PC (**Portability**).

iii. The system can create a backup database file after every transaction (sales, stock, service, update of authentication details) (**Recoverability**).

iv. For security issues only admin can change the password on behalf of staffs (**Integrity/Security**).

v. Staffs can only access this system for sales, service and checking reports(**Integrity/ Security**).

6. Use Case Diagram:

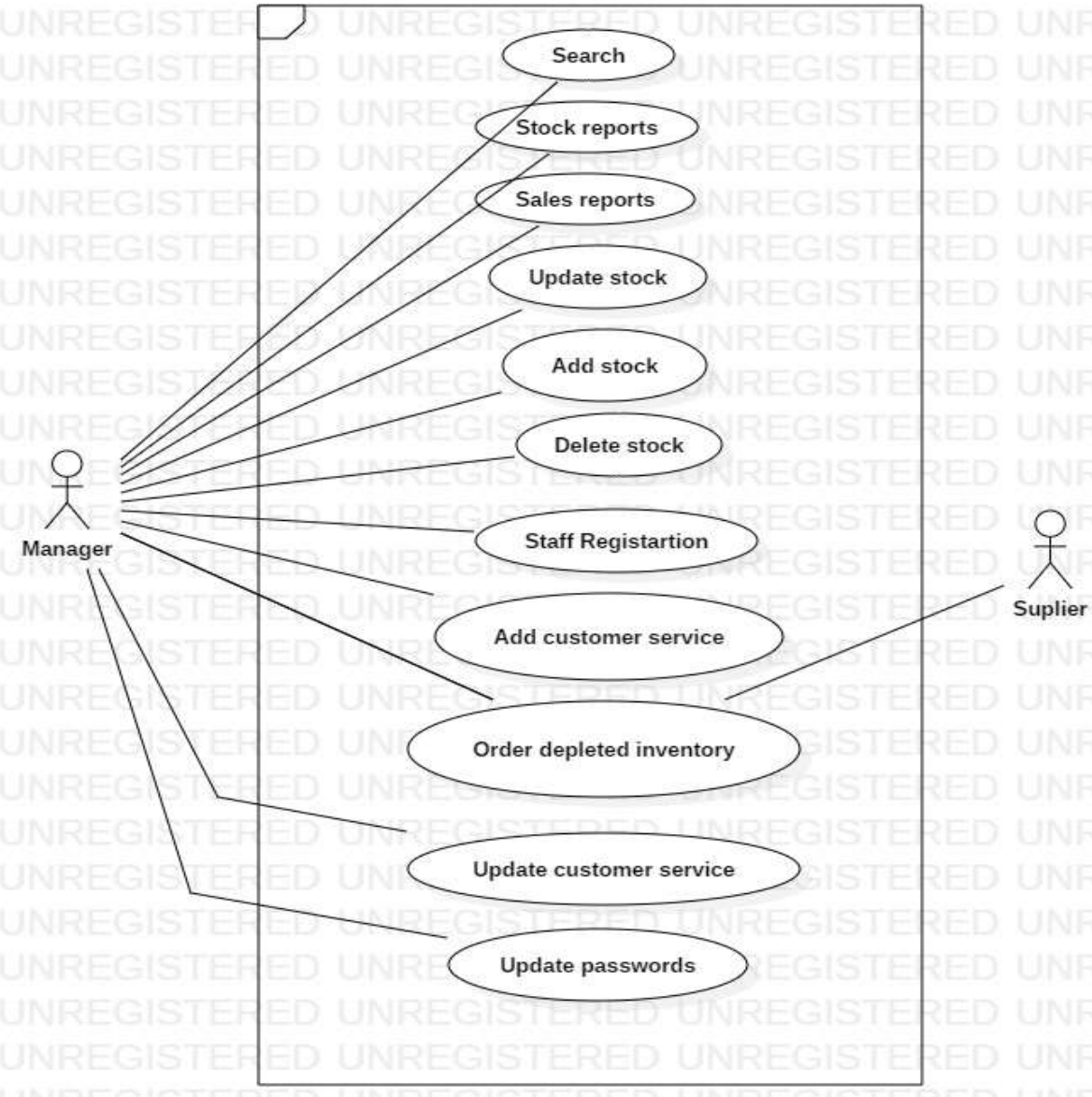


Fig. 6.1 Use Case diagram for Manager & Supplier

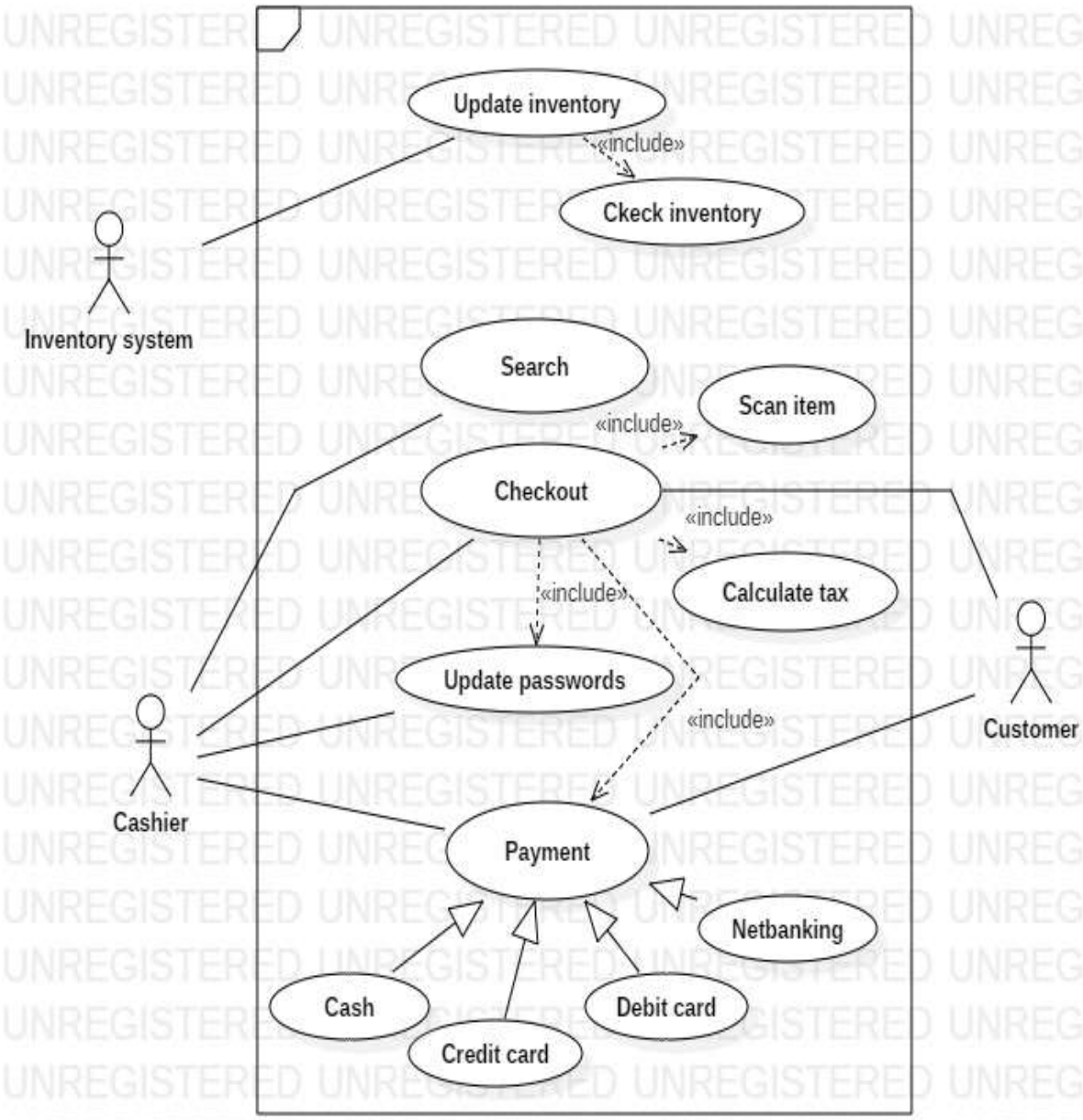


Fig. 6.2 Use Case diagram of Inventory system, Cashier and Customer

7. Sequence Diagram:

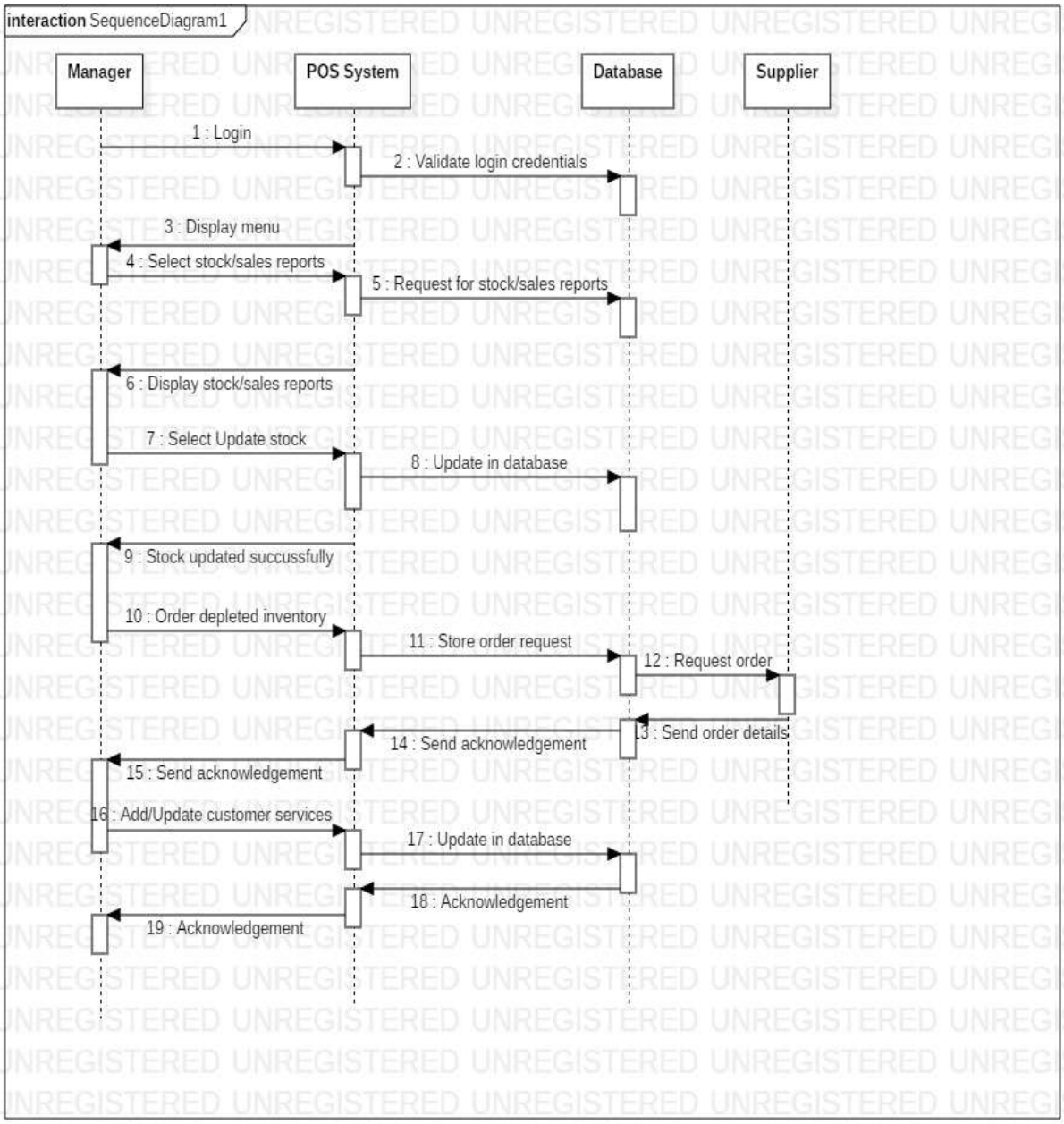


Fig.7.1 Sequence diagram for Manager and supplier

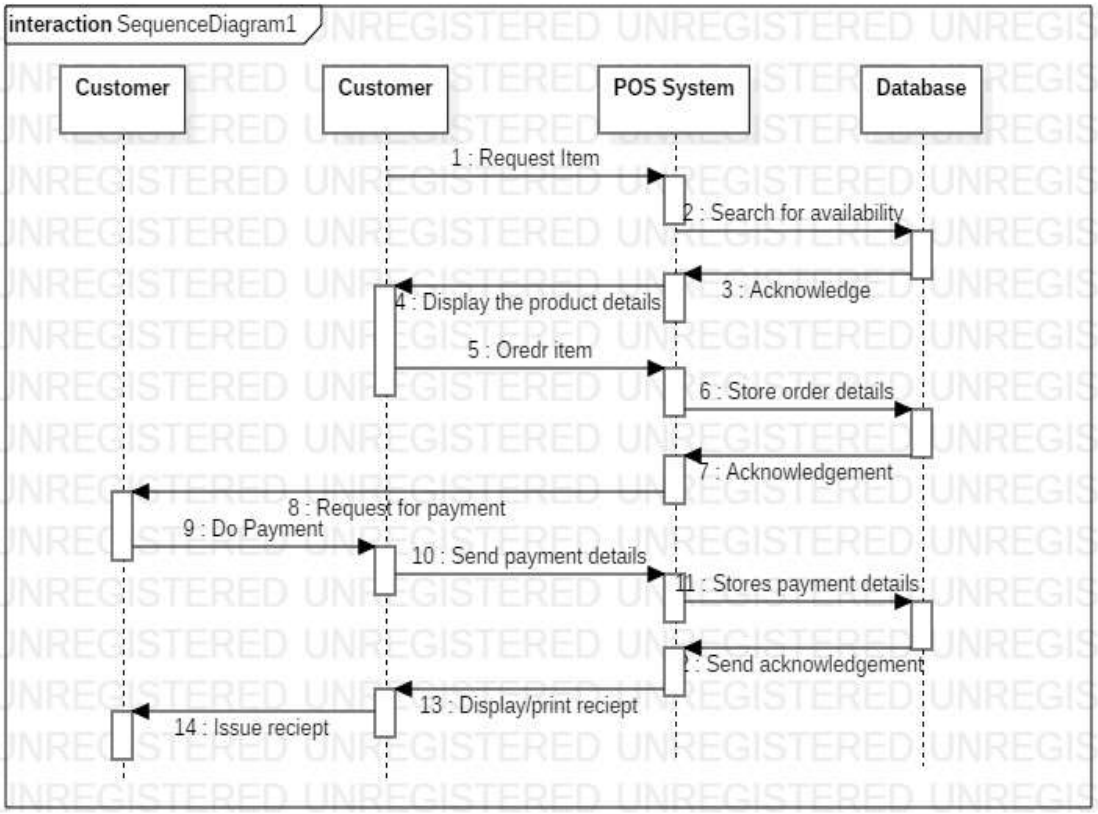
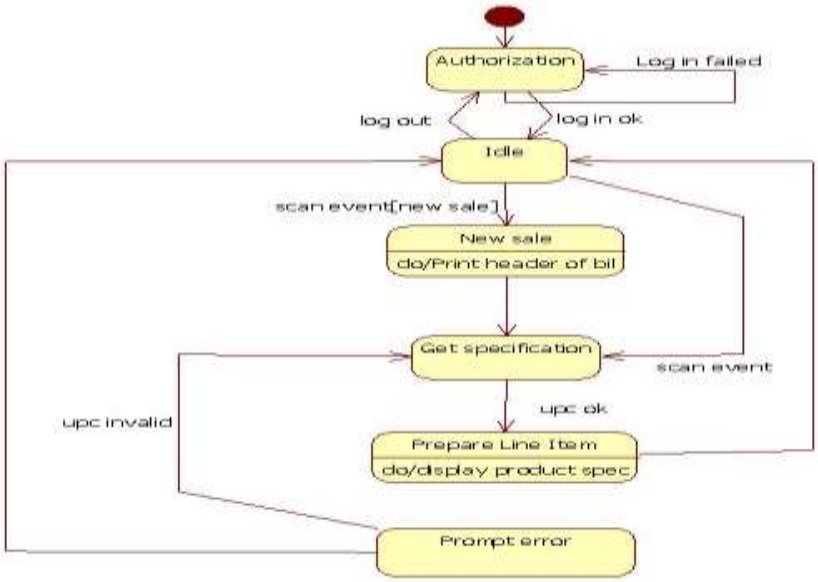
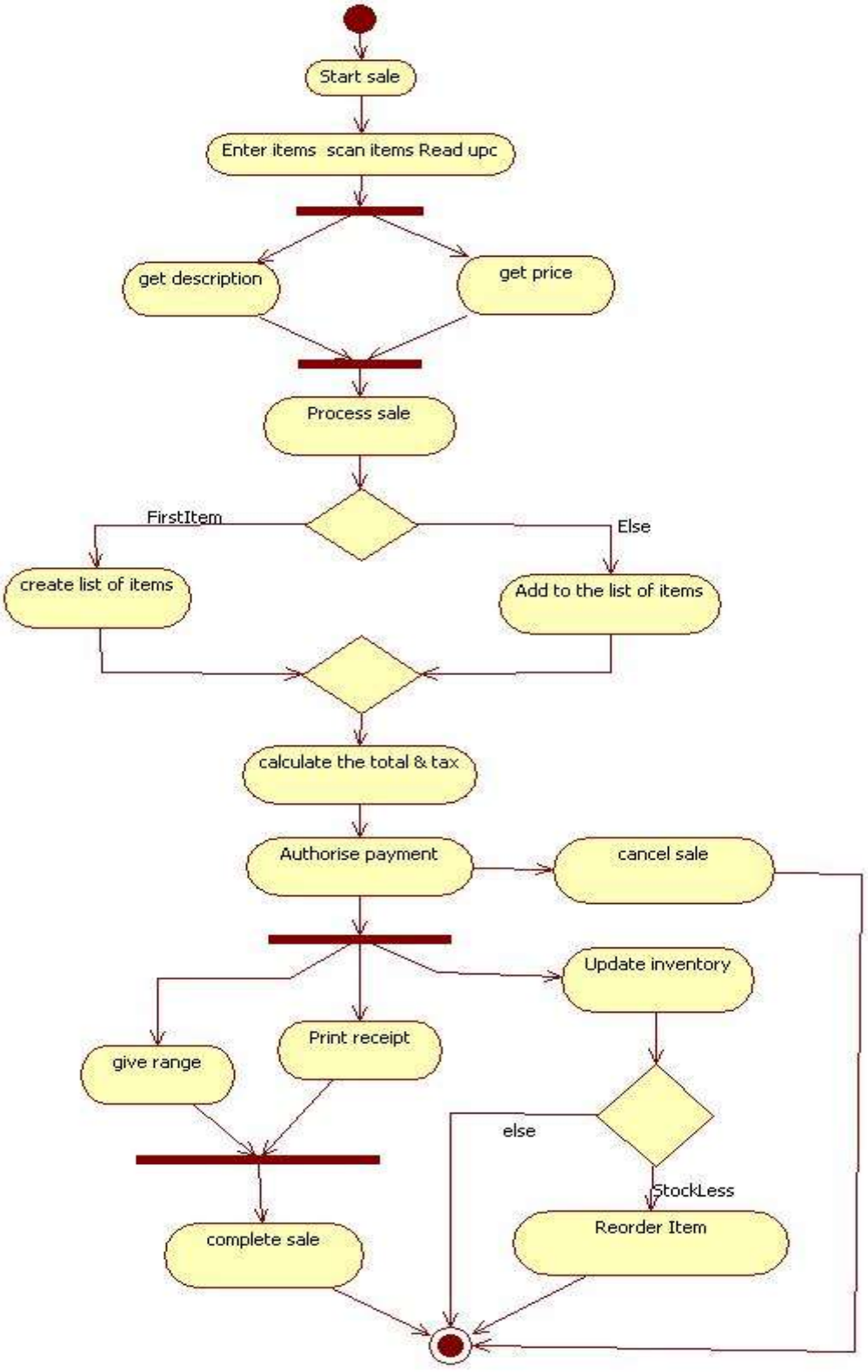


Fig.7.2 Sequence diagram for Customer and Cashier

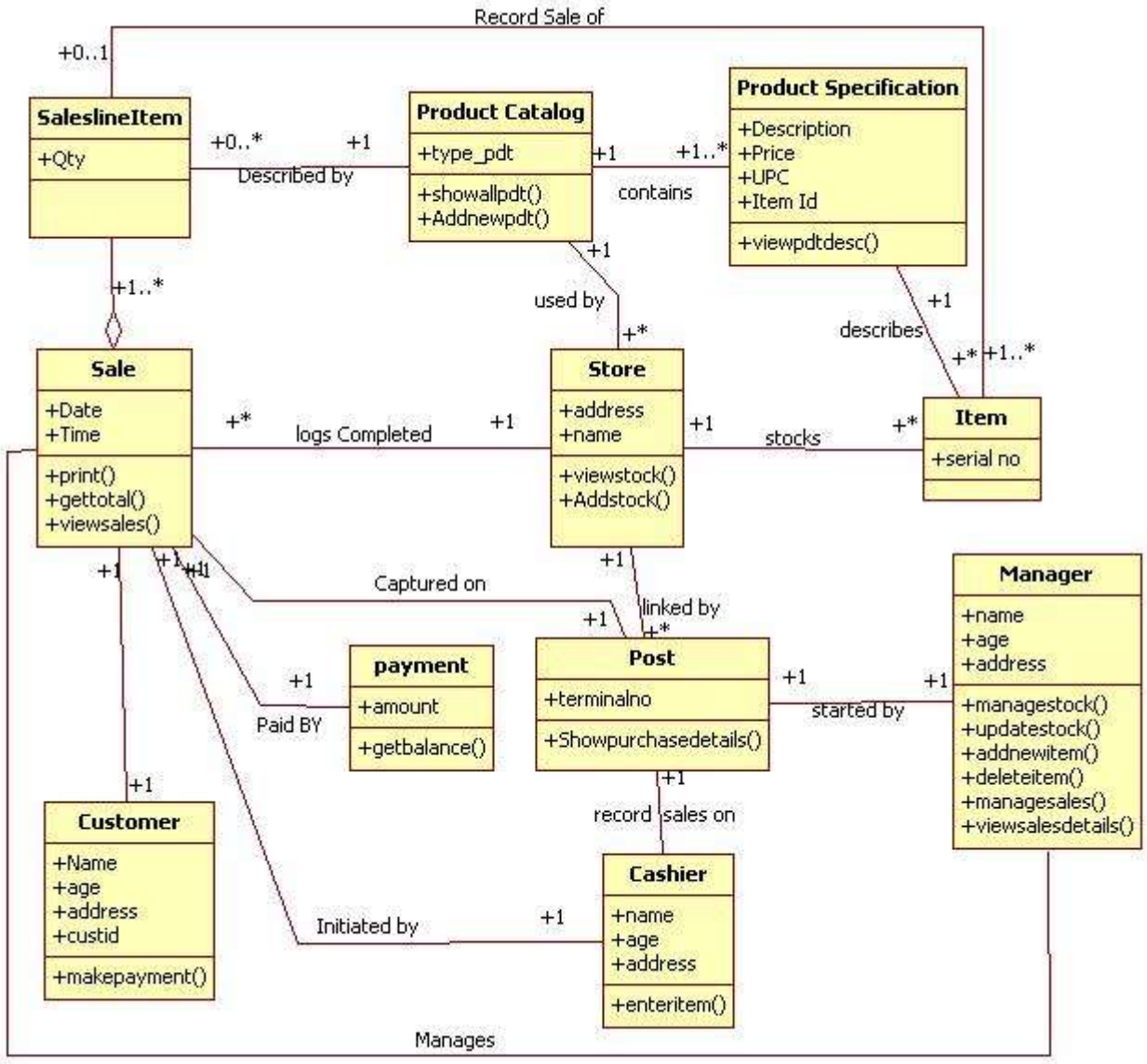
8. State Chart Diagram:



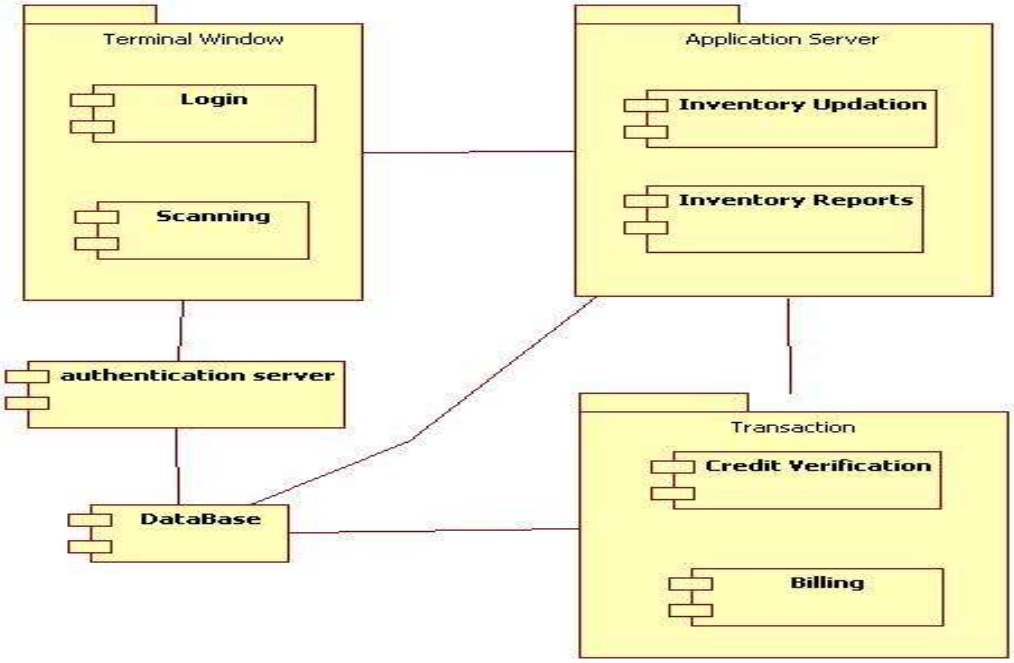
9. Activity Diagram:



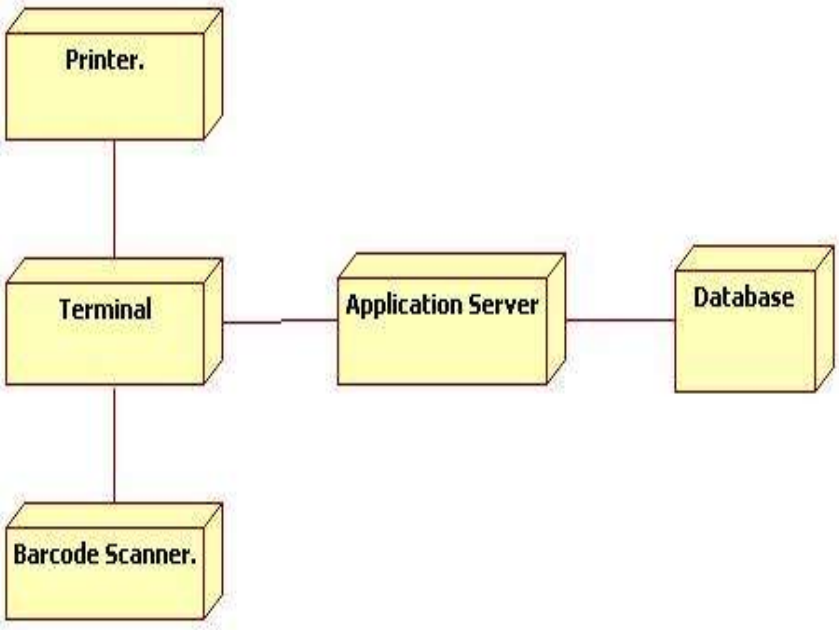
10. Class Diagram:


Record Sale of
+0..1
SaleslineItem
+Qty
+0..*
+1
Described by
Product Catalog
+type_pdt
+showallpdt()
+Addnewpdt()
+1
+1..*
contains
Product Specification
+Description
+Price
+UPC
+Item Id
+viewpdtdesc()
+1..*
used by
+1
+*
describes
+1
+*
+1..*
Sale
+Date
+Time
+print()
+gettotal()
+viewsales()
+*
logs Completed
+1
Store
+address
+name
+viewstock()
+Addstock()
+1
stocks
+*
Item
+serial no
+1
+1
+1
Captured on
+1
linked by
+*
Manager
+name
+age
+address
+managestock()
+updatestock()
+addnewitem()
+deleteitem()
+managesales()
+viewsalesdetails()
+1
Paid BY
payment
+amount
+getbalance()
Post
+terminalno
+Showpurchasedetails()
+1
started by
+1
+1
record sales on
+1
+1
Customer
+Name
+age
+address
+custid
+makepayment()
Initiated by
+1
Cashier
+name
+age
+address
+enteritem()
Manages

11. Component Diagram:



12. Deployment Diagram:



Project-II: Online Book Shop: Following the model of amazon.com or bn.com, design and implement an online bookstore.

2. Detailed Description: The main objective of the project is to create an online book store that allows users to search and purchase a book online based on title, author and subject. The selected books are displayed in a tabular format and the user can order their books online through credit card payment. Using this project the user can purchase a book online instead of going out to a book store and wasting time.

Online Book store is an online web application where the customer can purchase books online. Through a web browser the customers can search for a book by its title or author, later can add to the shopping cart and finally purchase using credit card transaction. The user can login using his account details or new customers can set up an account very quickly. They should give the details of their name, contact number and shipping address. The user can also give feedback to a book by giving ratings on a score of five. The books are divided into many categories based on subject like Software, Database, English, Architecture etc.

The Online Book Store Website provides customers with online shopping through a web browser. A customer can, create, sign in to his account, place items into a shopping cart and purchase using his credit card details.

The Administrator will have additional functionalities when compared to the common user. He can add, delete and update the book details, book categories,

member information and also confirm a placed order.

Advantages:

- Customers can get their book delivered instead of actually going and buying the book. They can make payment online itself.
- Managing of inventory in the shop for shopkeeper becomes easier as customers are not visiting and ordering online.
- This system saves both time and travelling cost of customers.
- User can get to know different kinds of books that they were unaware of by just searching in the system using keywords.

Disadvantages:

- The only disadvantage is if the customer receives a book that is not in proper condition or has some kind of defect then there incurs an additional charge of posting it back.

3. Actors:

There two actors who will use "Online Book Store" . They are

- Customer.
- Administrator.

Customer: He can, create, sign in to his account, place items into a shopping cart and purchase using his credit card details.

Administrator: He can add, delete and update the book details, book categories, member information and also confirm a placed order.

4. Functional Requirements :

This project has the following functionalities:

1) A Home page with product catalog

This is the page where the user will be navigated after a successful login. It will display all the book categories and will have a search keyword option to search for the required book. It also includes some special sections like recommended titles, weekly special books.

2) Search

A search by keyword option is provided to the user using a textbox .The keyword to be entered should be the book title.

3) Advanced Search

Advanced search helps the user to search for a book based on Title, Author, Category and price range. All the books which match the particular search criteria and their total count will be displayed .From here the user can select a book and add to the shopping cart.

4) Book Description

If the user would like to know details about a book he can click on the title from where he will be directed to a Book description page. It includes the notes on the book content and also a link to Amazon.com to get the book review.

5) User Voting

The user can give rating to a book based on his interest. He can rate it by giving a score of five as Excellent, four for very good, three for good, two for regular and one for deficient. The final rating of a book will depend on all the individual user rating.

6) Shopping Cart

The user can manage a shopping cart which will include all the books he selected. The user can edit, delete and update his shopping cart. A final shopping cart summary is displayed which includes all the items the user selected and the final total cost.

7) Managing user accounts

Each user should have an account to access all the functionalities of website. User can login using login page and logout using the logout page. All the user sessions will be saved in the database.

8) Administration

The Administrator will be provided with special functionalities like

- Add or delete a book category
- Add or delete a member
- Manage member orders.
- Add or delete a Credit Card type.

5. Non-Functional Requirements:

vi. The system can save stock into the database safely.

vii. The system can support all the PC (Personal Computer).

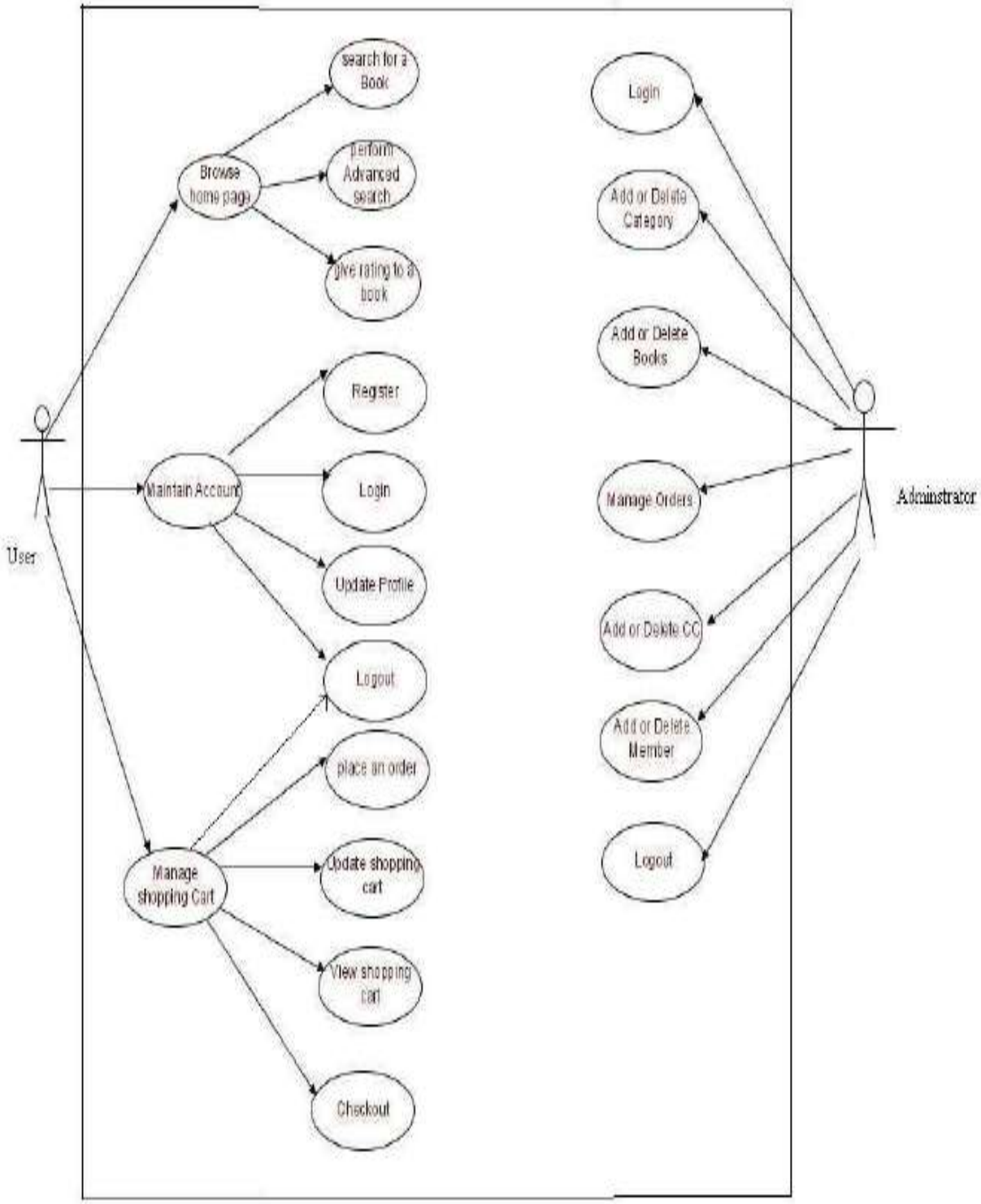
viii. The system can create a backup database file after every transaction (sales, stock, service, update of authentication details).

ix. Stock should be added after end of sales per day.

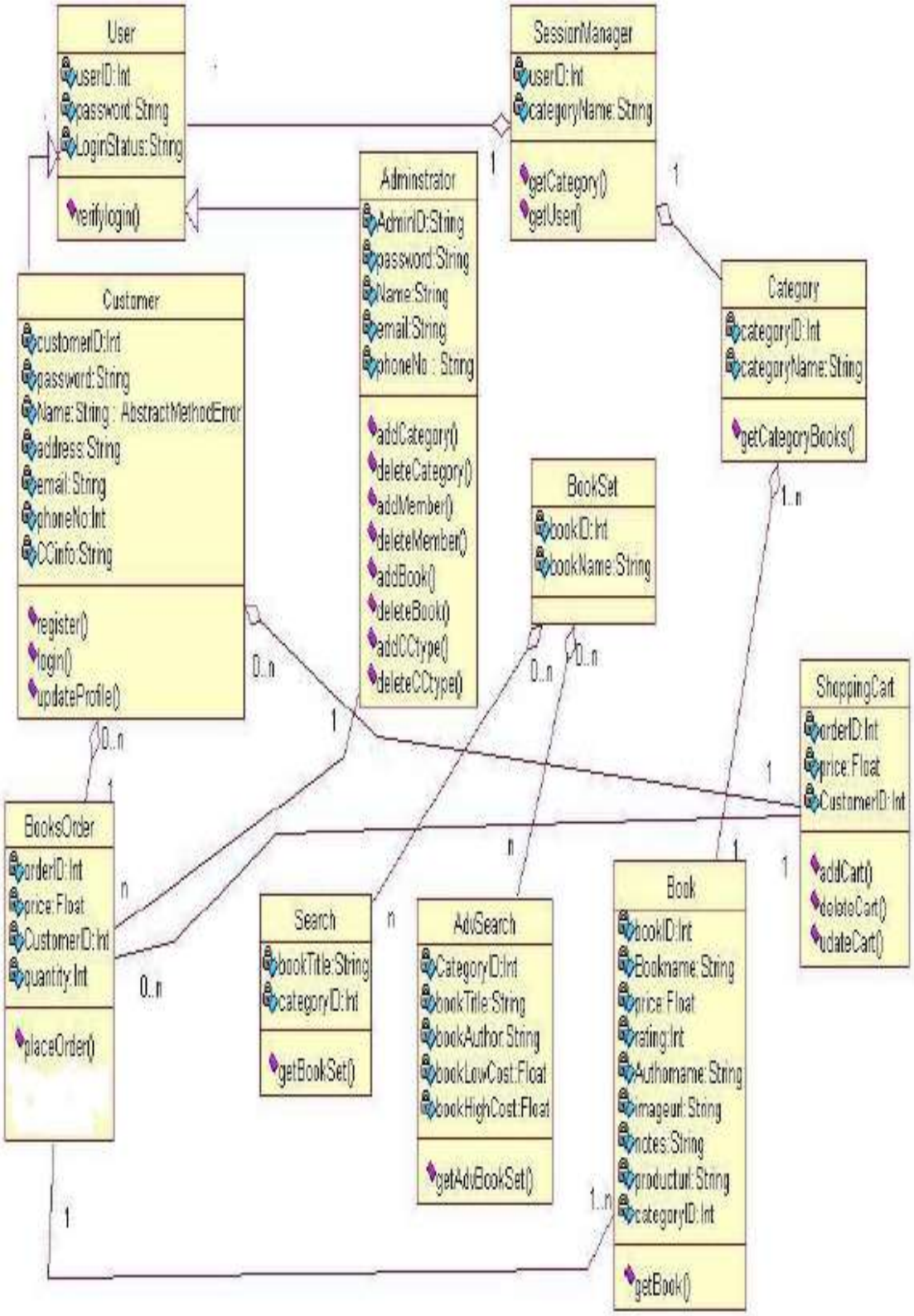
x. For security issues only admin can change the password on behalf of staffs.

xi. Staffs can only access this system for sales, service and checking reports.

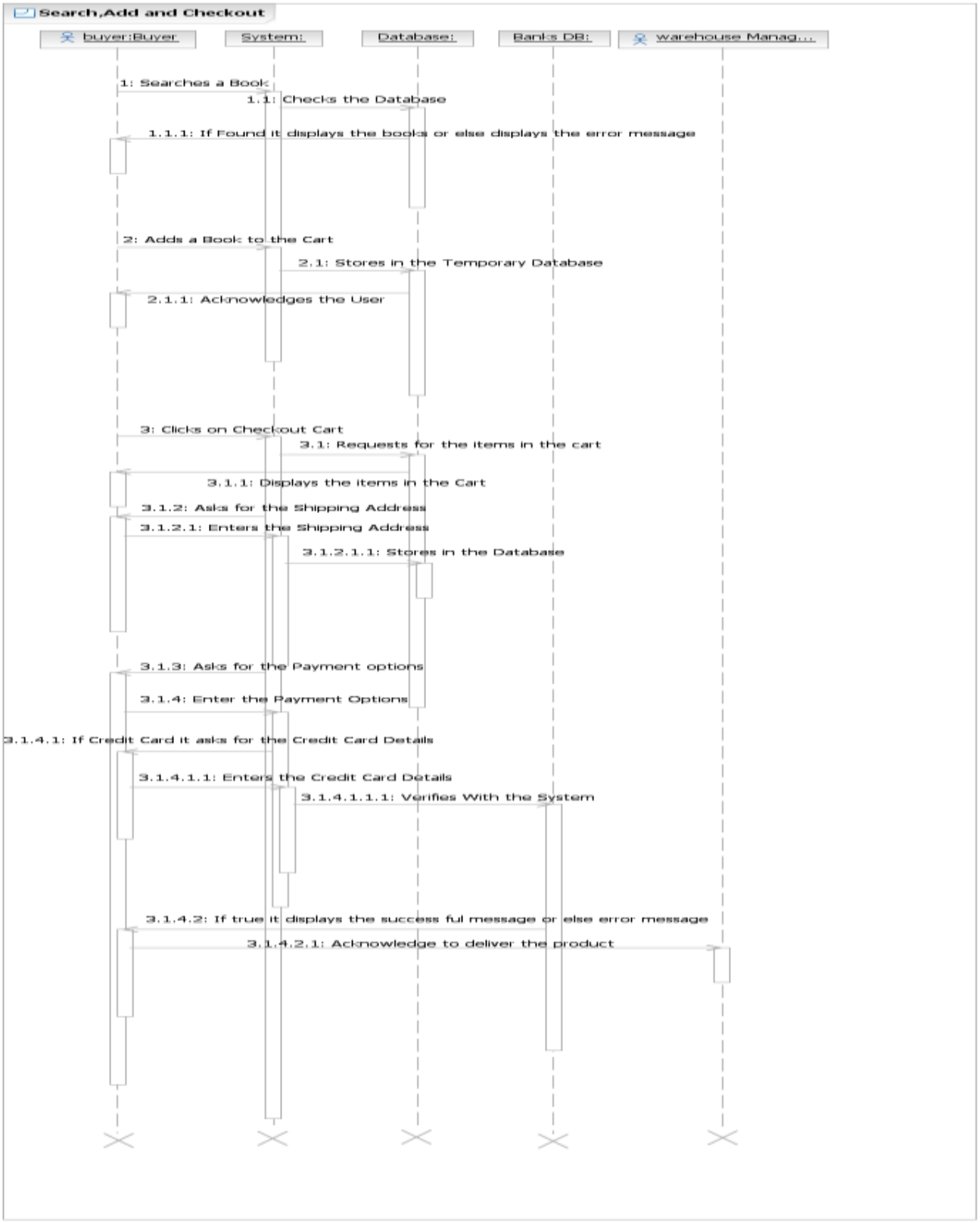
6. Use Case Diagram:



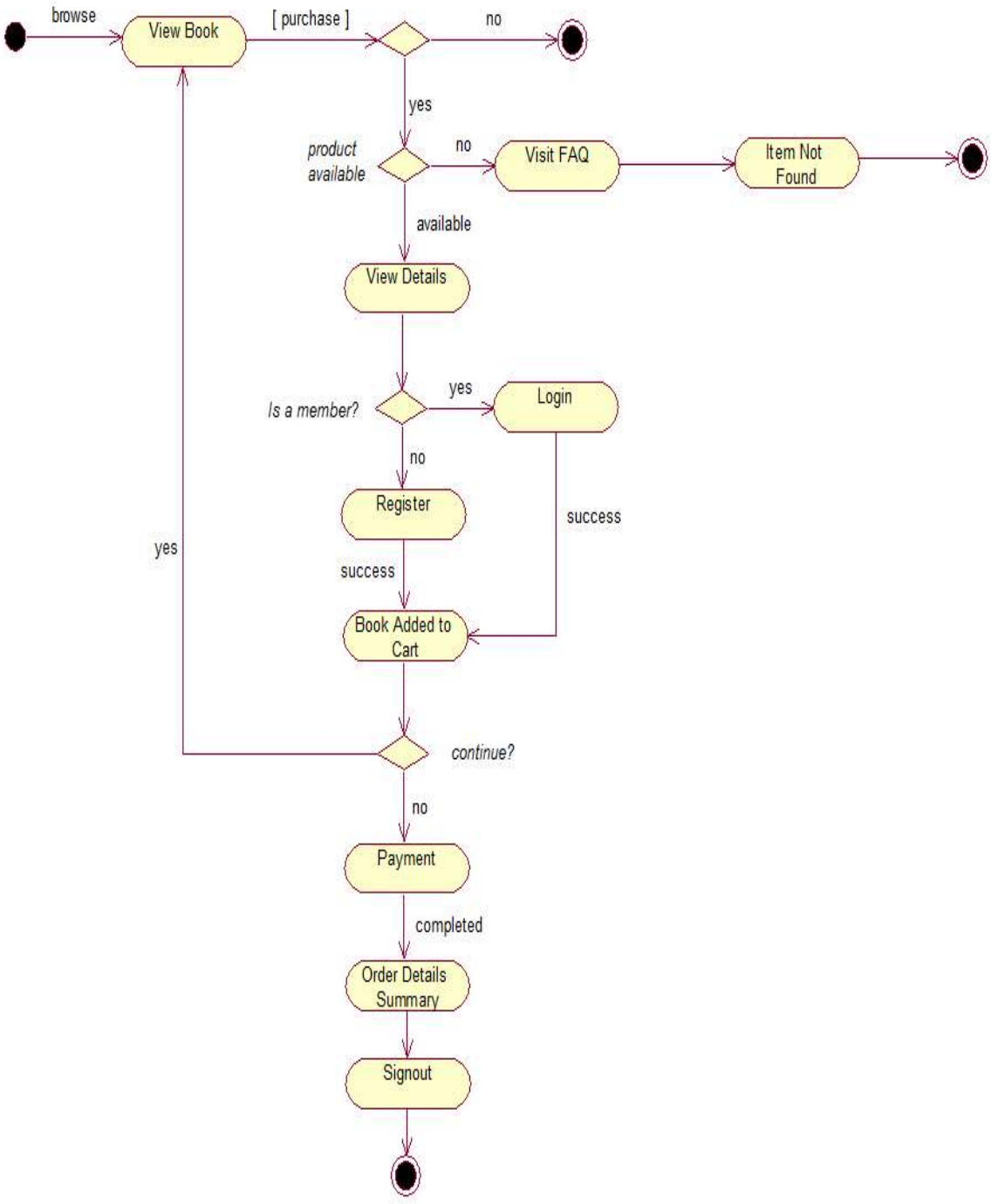
7. Class Diagram:



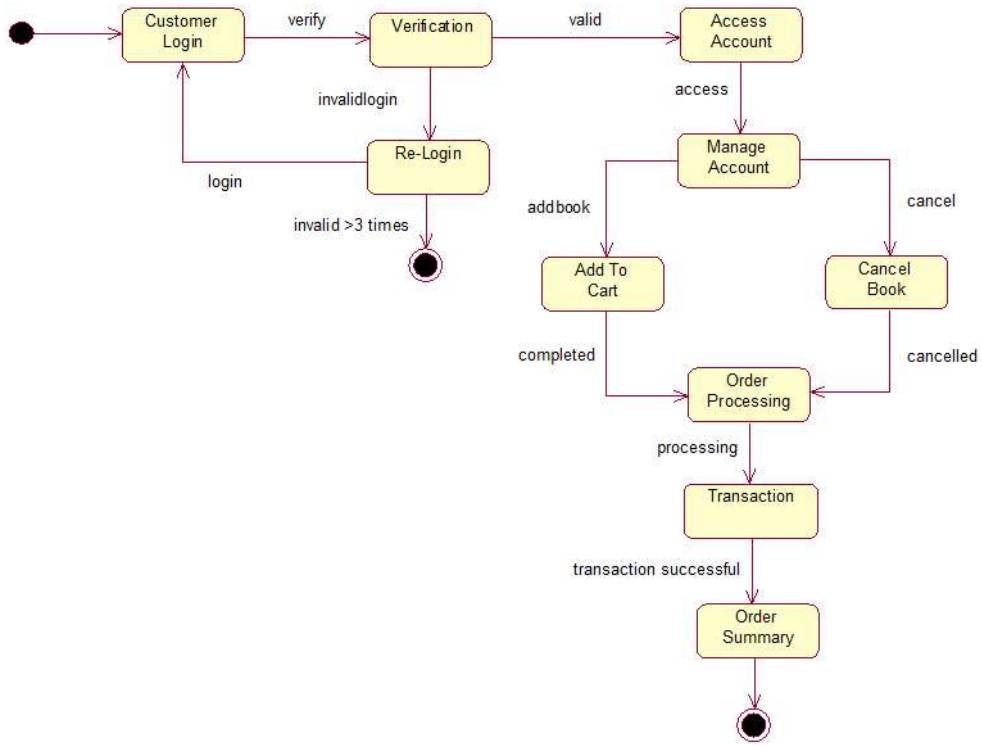
8. Sequence Diagram:



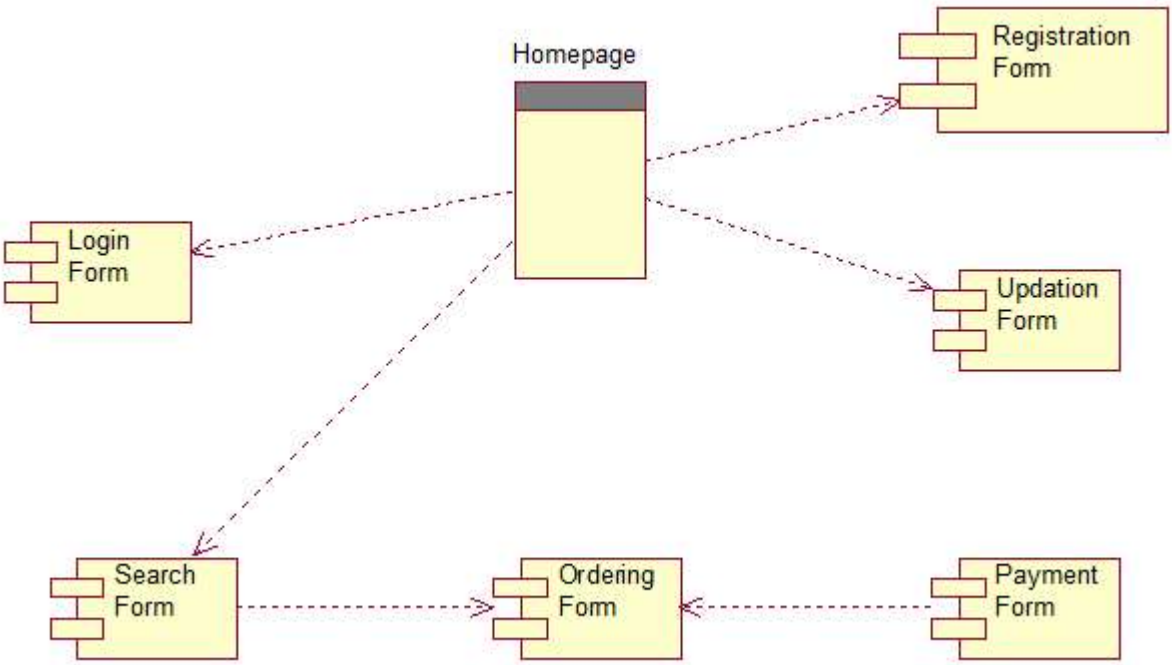
9. Activity Diagram:



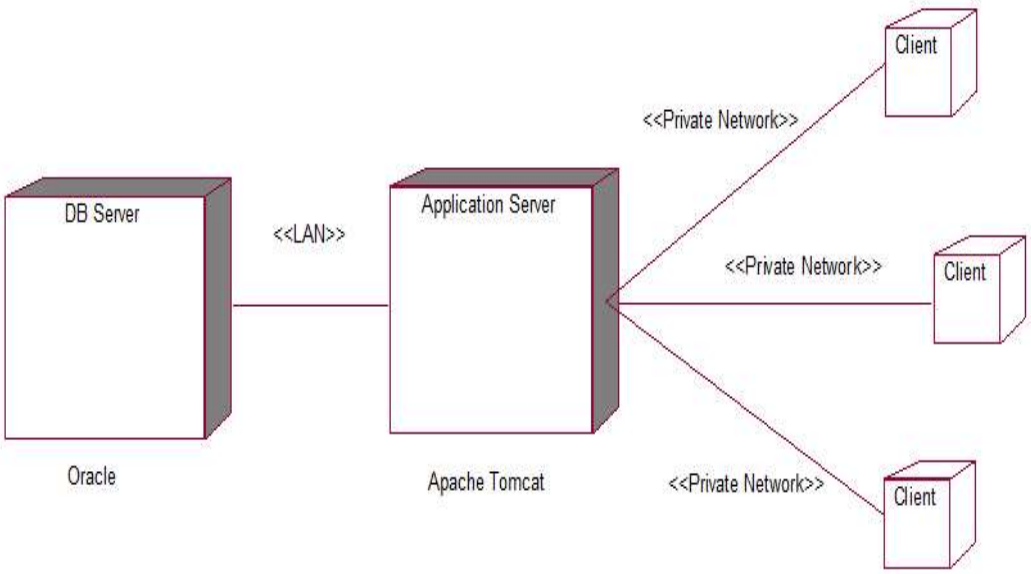
10. State Chart Diagram:



11. Component Diagram:



12. Deployment Diagram:



Project-III: Simulated Company: Simulate a small manufacturing company. The resulting application will enable the user to take out a loan, purchase a machine, and over a series of monthly production runs, follow the performance of their company.

2. Brief Description about Simulated Company: This project is aimed to design the modeling and simulation infrastructure to experiment and to validate the proposed solutions. It is an example of documents produced when undertaking the analysis and design of an application which simulates the small manufacturing company.

This project is mainly focused on the user to take a place, purchase the machinery and over a series of monthly and yearly production runs follows the concept of the company. The company has to see all the profits and losses. They have to see all dealings of the company and see the additional features of the machine for better development. The company accounts are updated for a given month. The accounts are considered to compute gross profits from the sales. General expenses such as salary and the rent are taken into account to calculate the net profit of the company. In addition to these details, inventory and sales are required to be updated.

Basic considerations or assumptions in the design of the project:

- ✓ The company is supposed to take the loan and repay the loan.
- ✓ Company is proposed to purchase the machinery and start the production.
- ✓ The sales manager will look after the goods sales and update the details in the stock record.
- ✓ Based on the record statistics, performance of the department is evaluated.

3. Actors:

There are various actors who will look after the simulated company and they are

Founder Chairman: He is the founder chairman of the company. He will be the liable person for the development of the company.

Bank: Banks are intended to issue the loan and keep on evaluate the performance of the company.

Manager is an actor, who is intended to look after the following departments.

Purchase Department: This department will purchase the machines and then submit the bills to the authorities.

Production Department: This department will manufacture the products and sends the details to the company.

Marketing Department: This department will look after the sales of the goods produced by the production department. Later, these people will update the statistics in the records.

Performance analysis department: This department will assess the performance of the company on regular basis and identifies the status of the company as like in profit or in loss.

For the sake of simplicity, assuming that Chairman is going to act the manager role of all the departments.

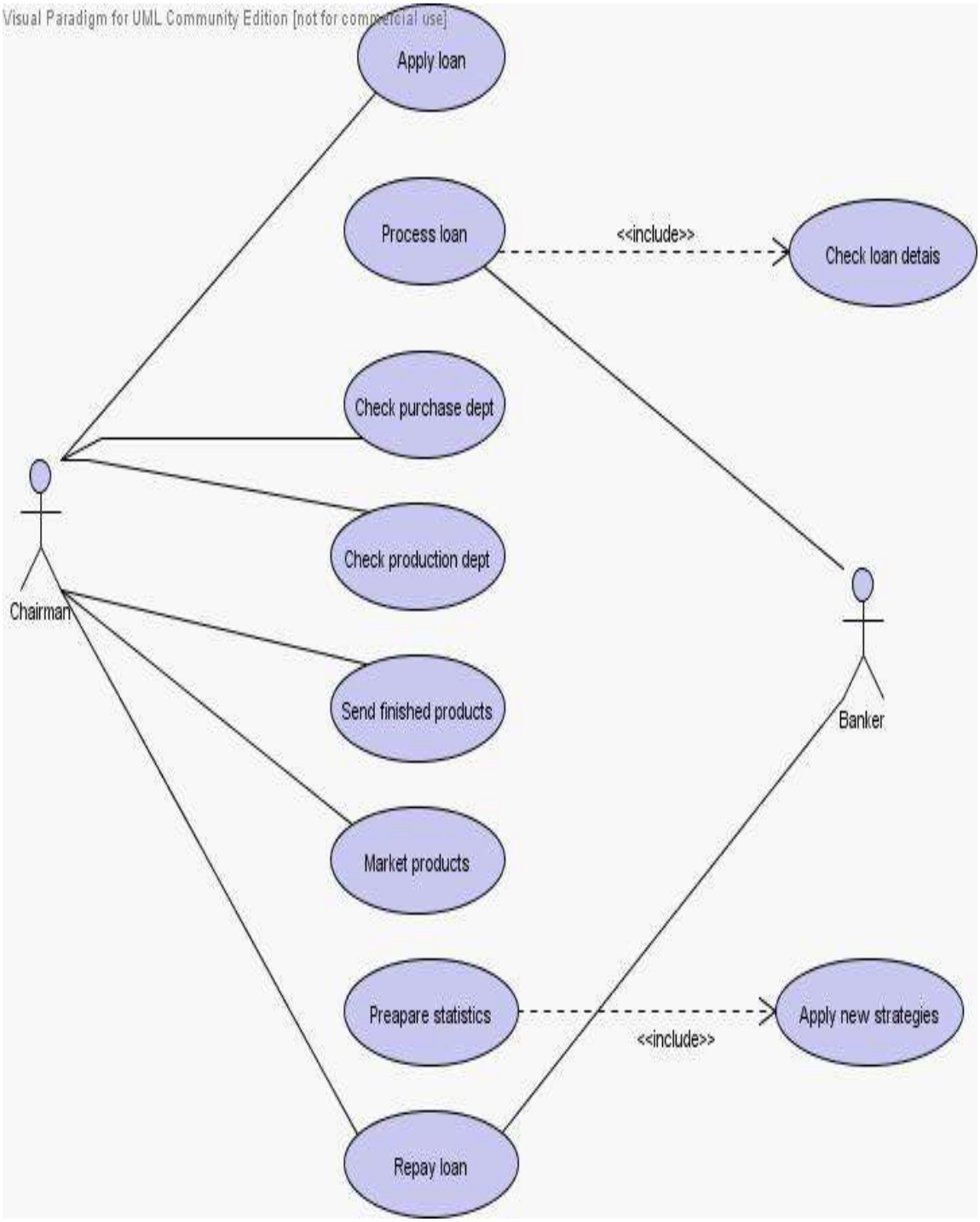
4. Functional Requirements:

- ✓ Loan Processing includes application, processing and issuing etc.
- ✓ Performance evaluation of the individual departments like production, marketing etc.
- ✓ Based on the record statistics, the current status of the company is required to be evaluated.
- ✓ Banks are intended to give the loans based on the suitability and availability.
- ✓ Based on the gross profits and loss, companies are expected to repay the loans.

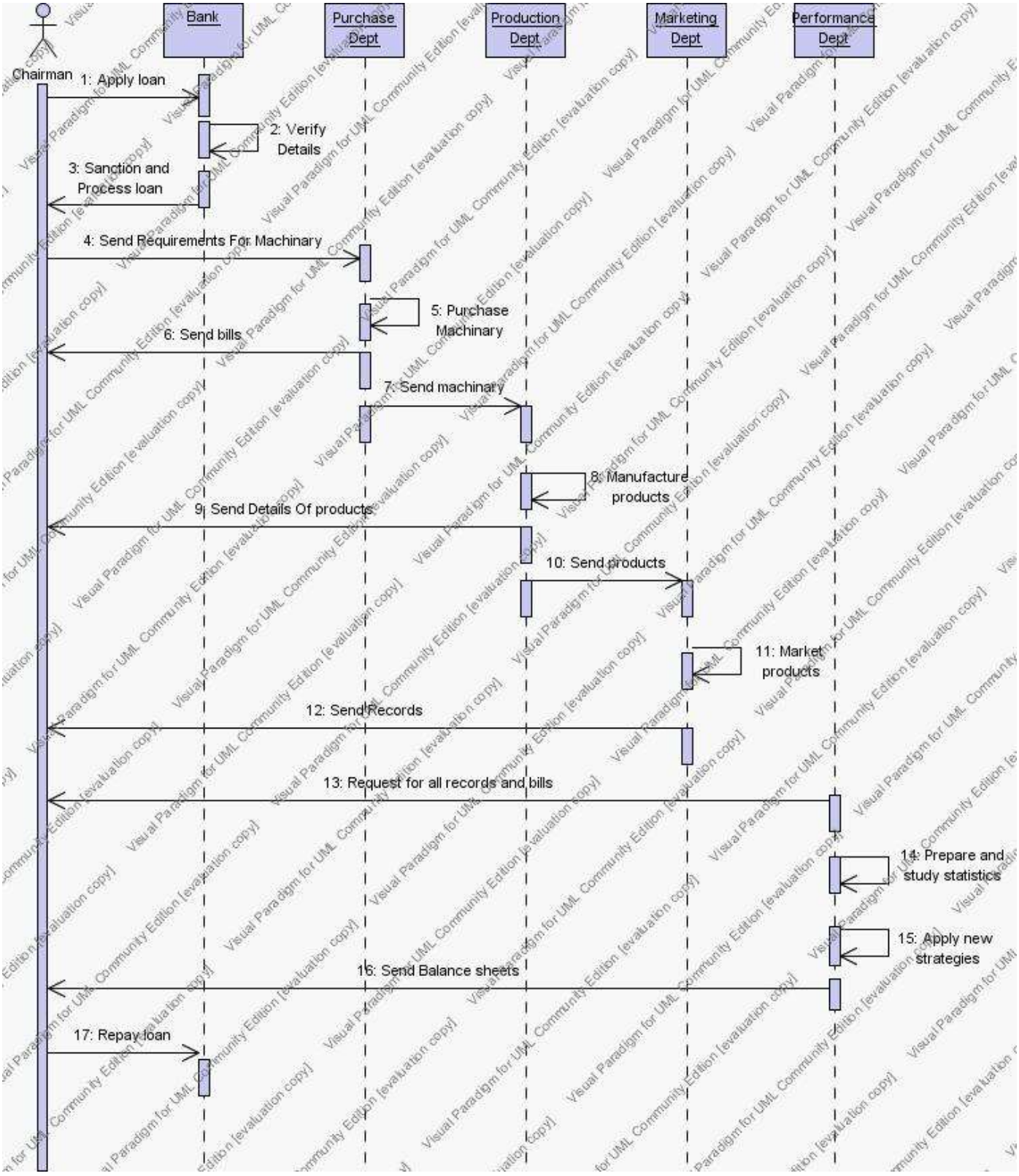
5. Non-Functional Requirements:

- ✓ Adaptability
- ✓ Compatibility
- ✓ Transparency
- ✓ Usability
- ✓ Integrity
- ✓ Fault tolerance

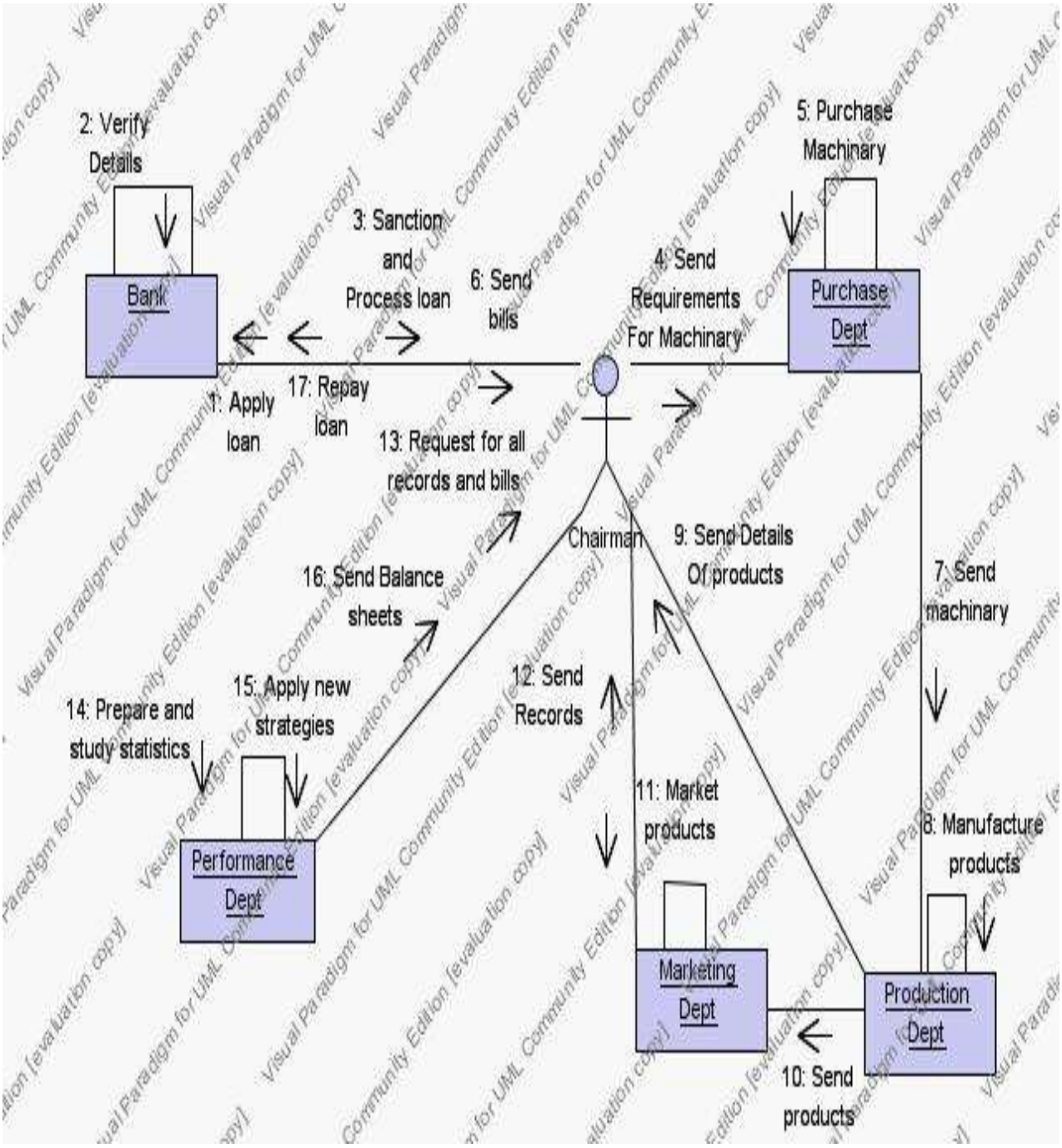
6. Use Case Diagram:



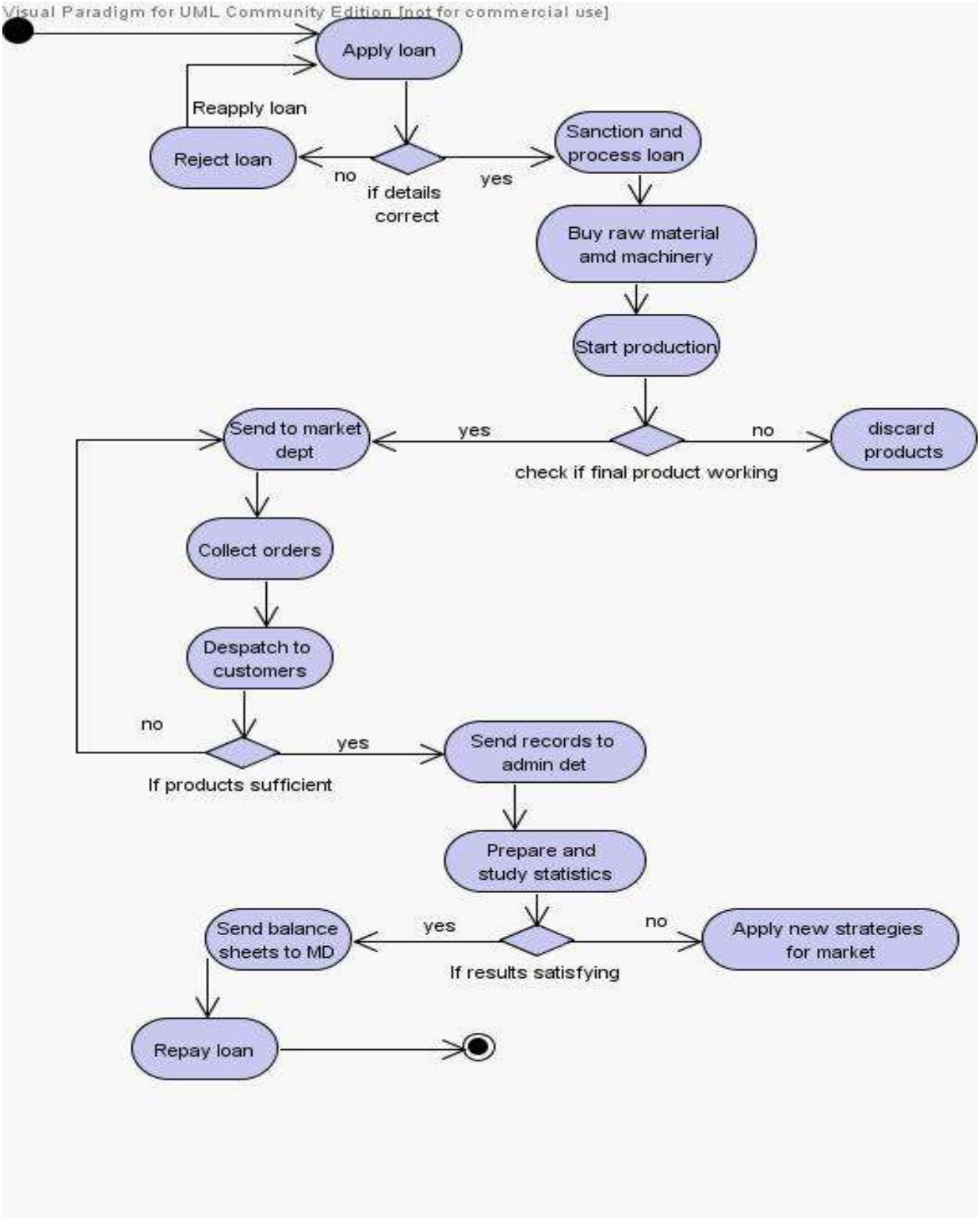
7. Sequence Diagram:



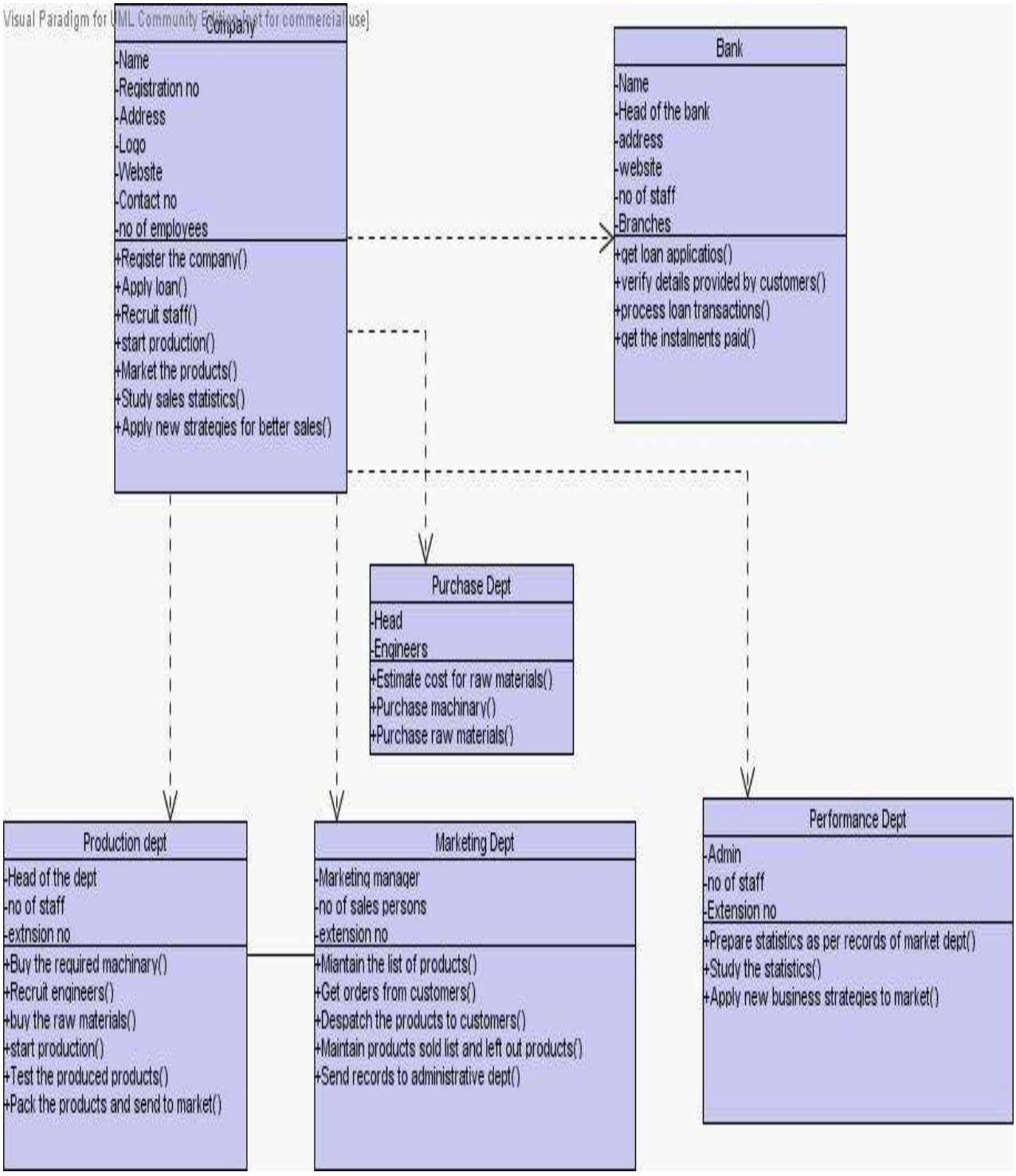
8. Collaboration Diagram:



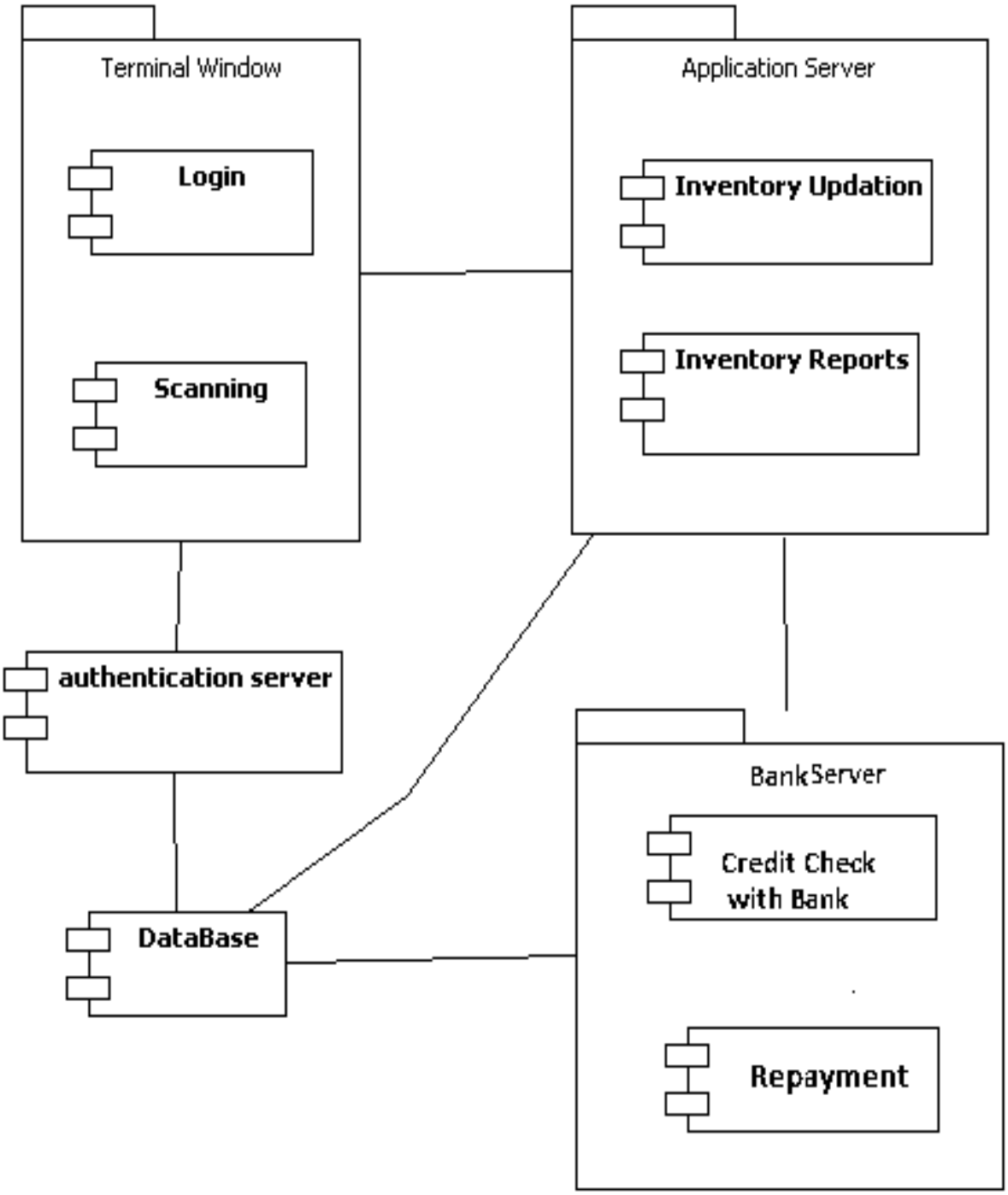
9. Activity Diagram:



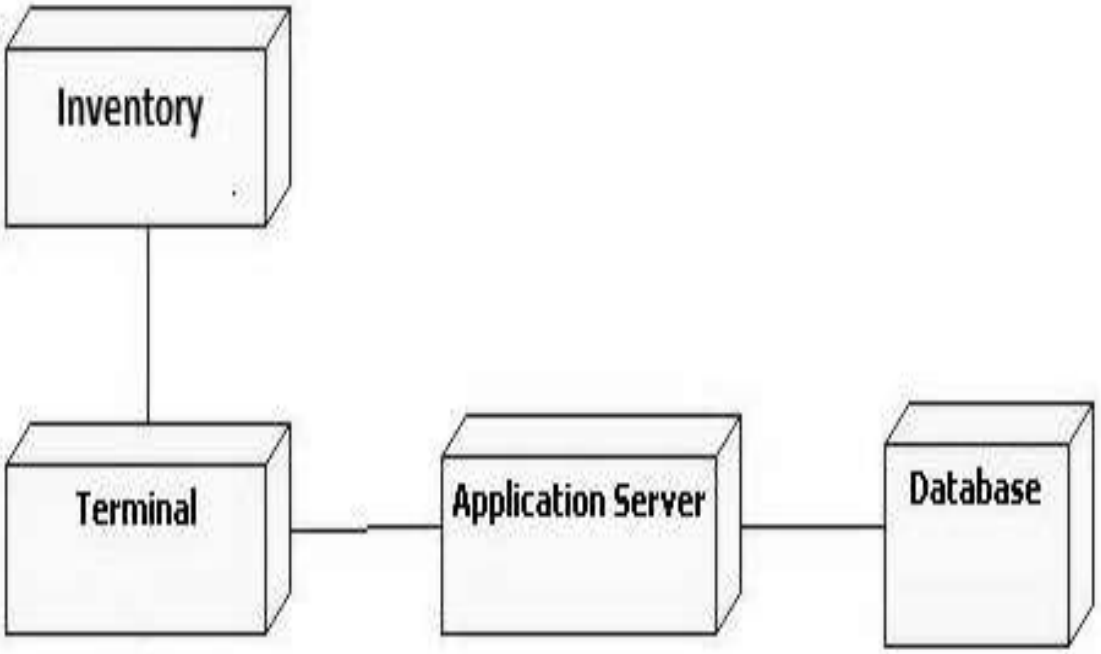
10. Class Diagram:



11. Component Diagram:

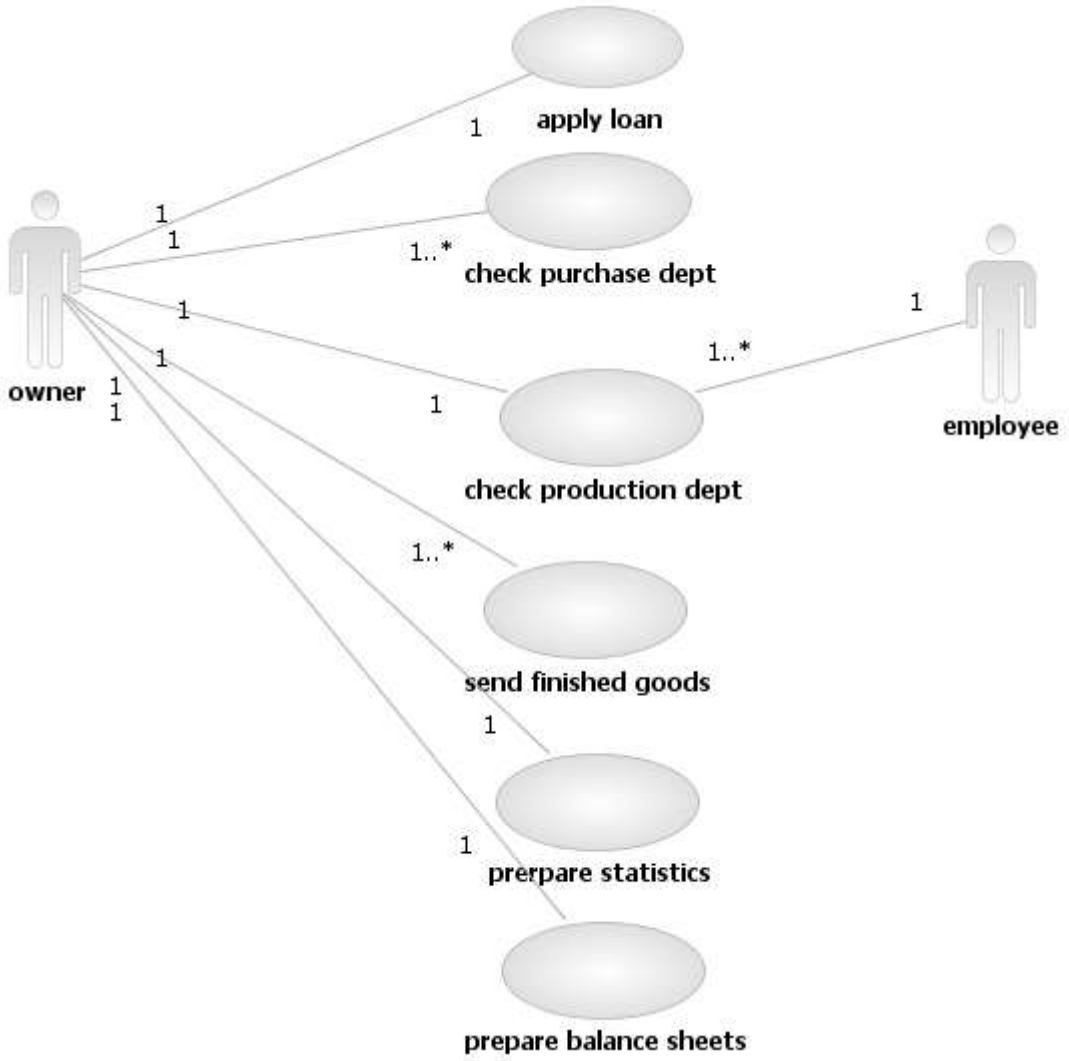


12. Deployment Diagram:

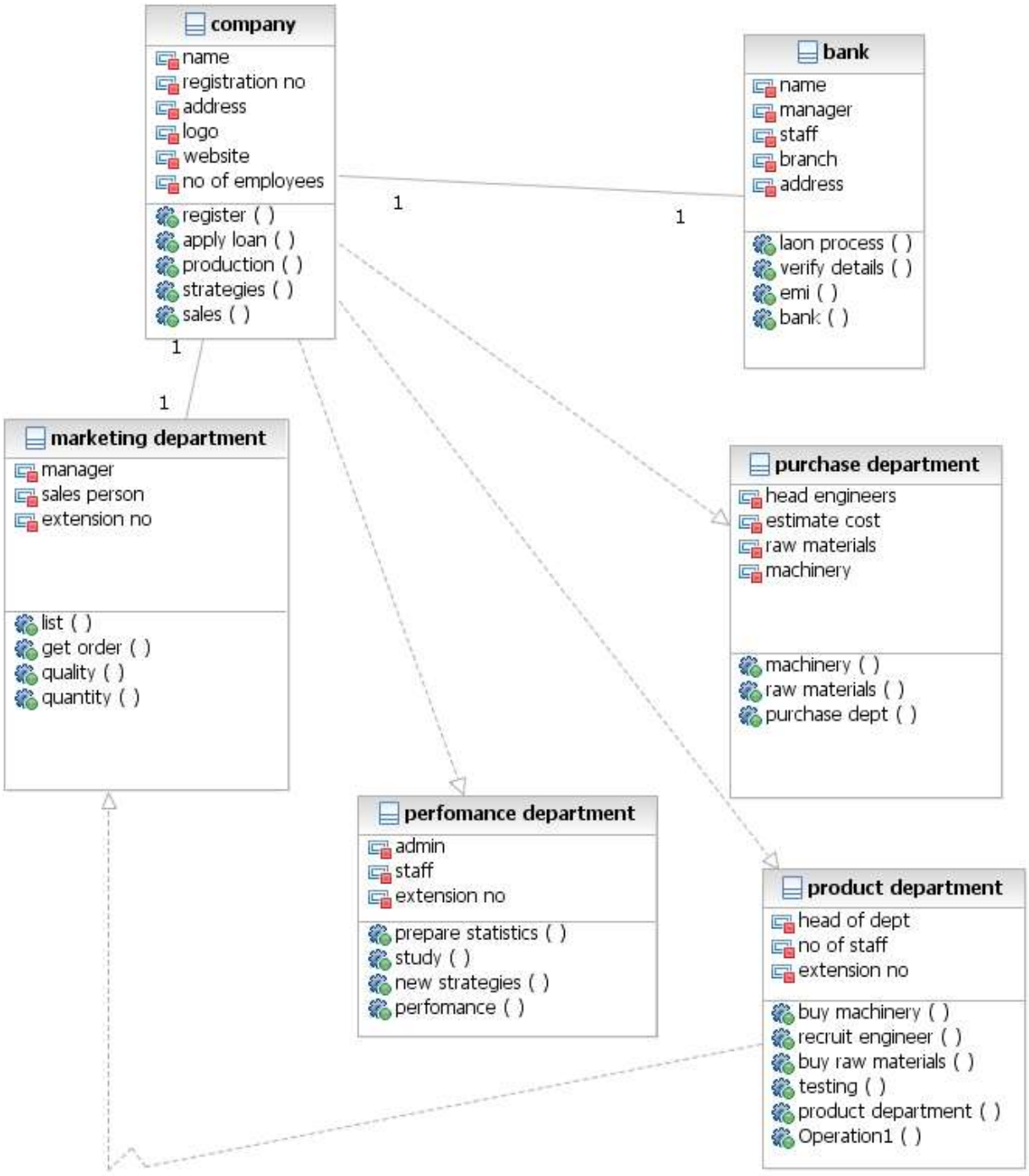


Project-III- A Simulated Company : Simulate a small manufacturing company. The resulting application will enable the user to take out a loan, purchase a machine, and over a series of monthly production runs, follow the performance of their company.

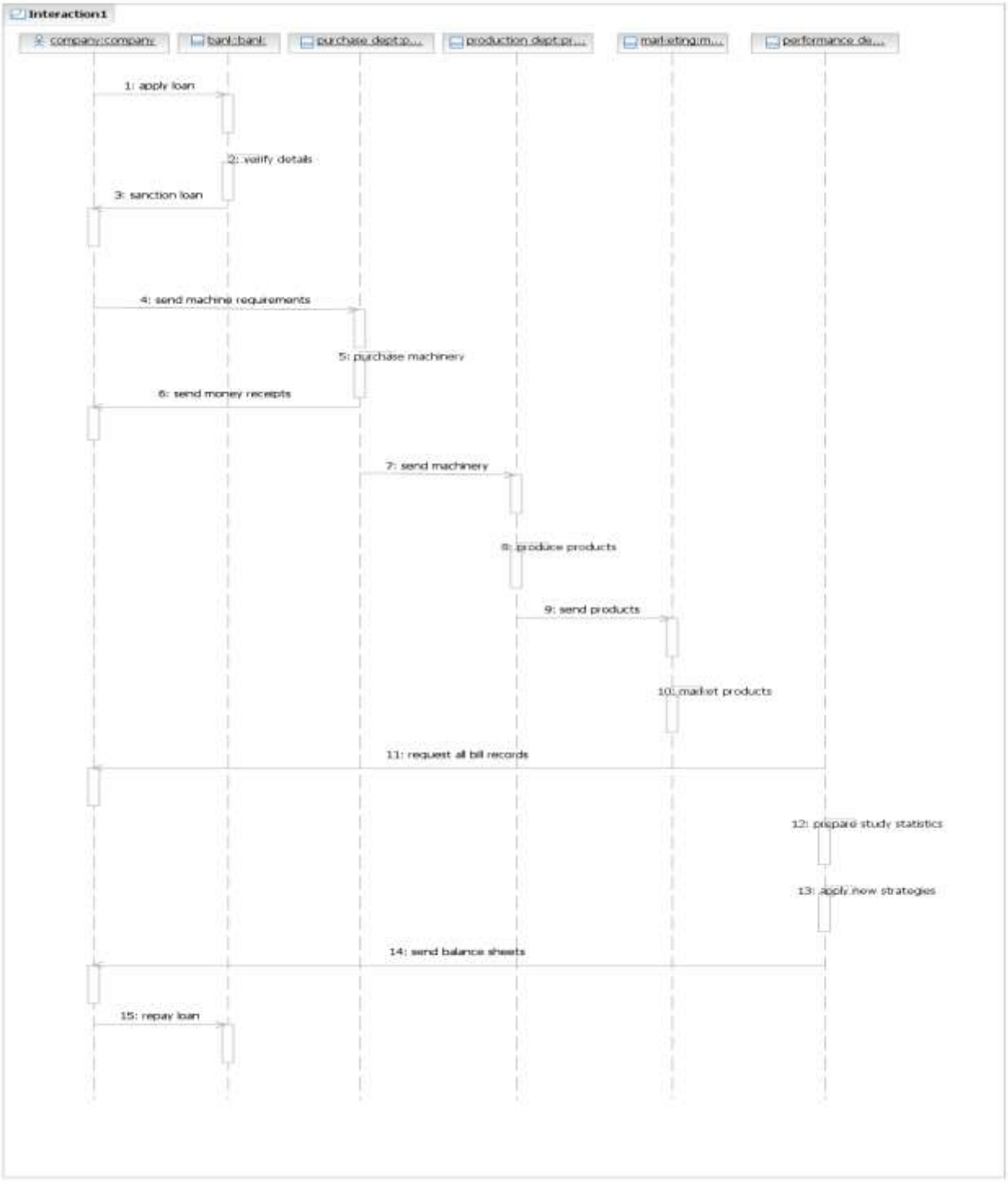
Usecase Diagram



Class Diagram



Sequence Diagram



Project-IV: Multi-Threaded Airport Simulation System: A critical step of the project is to design a modeling and simulation infrastructure to experiment and validate the proposed solutions

The ever growing demand of air transport shows the vulnerability of the current air traffic management system: Congestion, time delays, etc., particularly in poor weather conditions.

The project is focused on controller and pilot assistance systems for approach and ground movements. The critical step of the project was to design an airport modeling and simulation infrastructure to improve the safety and efficiency of ground movements in all weather conditions. It simulates the arrivals and departures at an airport in a time sequence. During every minute, planes may enter the systems, they may land, they may take off, or they may crash. The project must keep track of planes, assign planes to runways, execute the take offs and landings, and keep track of status of each plan, runway and terminal. So the finally made computer software should model various aspects of the total airports operation-connecting airside and landside, literally from the airspace to the curb.

As part of case study, following analysis diagrams will be created

1. Use cases for the system.
2. Class diagram for initially identified classes.
3. Activity diagram to show flow for each use case.
4. Sequence and Collaboration diagram.
5. State chart diagram shows states before and after each action.

Conceptualization

Assumptions;

- All take offs take the same amount of time and all landings take the same amount of time (through these two times may be different).
- Planes arrive for landing at random times, but with a specified probability of a plane arriving during any given minute.

- Planes arrive for takeoff at random times, but with a specified probability of a plane arriving during any given minute
- Landings have priorities over takeoffs.

- Planes arriving for landing have a random amount of fuel and they will crash if they do not land before they run out of fuel.

Input will be:

- The amount of time needed for one plane to land.
- The amount of time needed for one plane to takeoff.
- The probability of a plane entering the landing queue in any given minute.
- The probability of a plane entering the takeoff queue in any given minute.
- The maximum minutes until a plane waiting to land will crash.
- The statues of each runway, plane and terminal.

The Output of the program will be:

- Total simulation time.
- The number of planes that takeoff in the simulated time.
- The number of planes that landed in the simulated time.
- The average time a plane spent in the takeoff queue.
- The average time a plane spent in the landing queue.
- Updated status of each runway, plane, and terminal.

Requirement Analysis:

Textual Analysis:

This covers the requirements and diagrams of the project. The complete simulation of airport control system as follows

Actors:

These are who are involved in interaction of the whole process.

1. **Technical head:** He is the person who supervises the controls the ground traffic on runway. He checks the status of runways and assigns the free runways and terminals for takeoff and landing.
2. **Pilot:** He is the person who controls the aircraft. He transmits or receives signals regarding the free runways, and terminal from the control room. He is responsible for the safe landing or takeoffs the planes.

Use cases:

The steps involved in the whole process are indicated as use cases.

- Transmit/receive signals.
- Check availability of runways.
- Land the plane.
- Check if time left for next departure.
- Check for free terminal.
- Update status of runway, terminal.

1. **Transmit/receive signals:** The pilot in the aircraft transmits signals for requesting a free runway to takeoff or land. The control room on the ground receives these signals from the aircrafts.
2. **Check availability of runway:** The status of each runway in the airport is checked if it's free and its going to be free until the particular aircraft is landed or takeoff. If this is going to be free then runway number is transmitted to the pilot on aircraft.
3. **Land the plane:** The plane is landed safely on the airport as per directions given by the control room regarding runway and timings.
4. **Check if time left for next departure:** If the plane leaves immediately after landing then assign again a runway for takeoff. If there is still time then the plane has to be parked in a terminal.
5. **Check availability of terminals:** the status of each terminal is to be checked to find a free terminal. It should be checked whether that particular model of plane fits into that terminal. Then that particular terminal has to be assigned to the plane.

6. **Update Status:** the status of runway and terminal are to be set to be using while using them. The status has to be immediately changed as soon as the work is complete. This should be supervised carefully to avoid collisions and crashes of aircrafts.

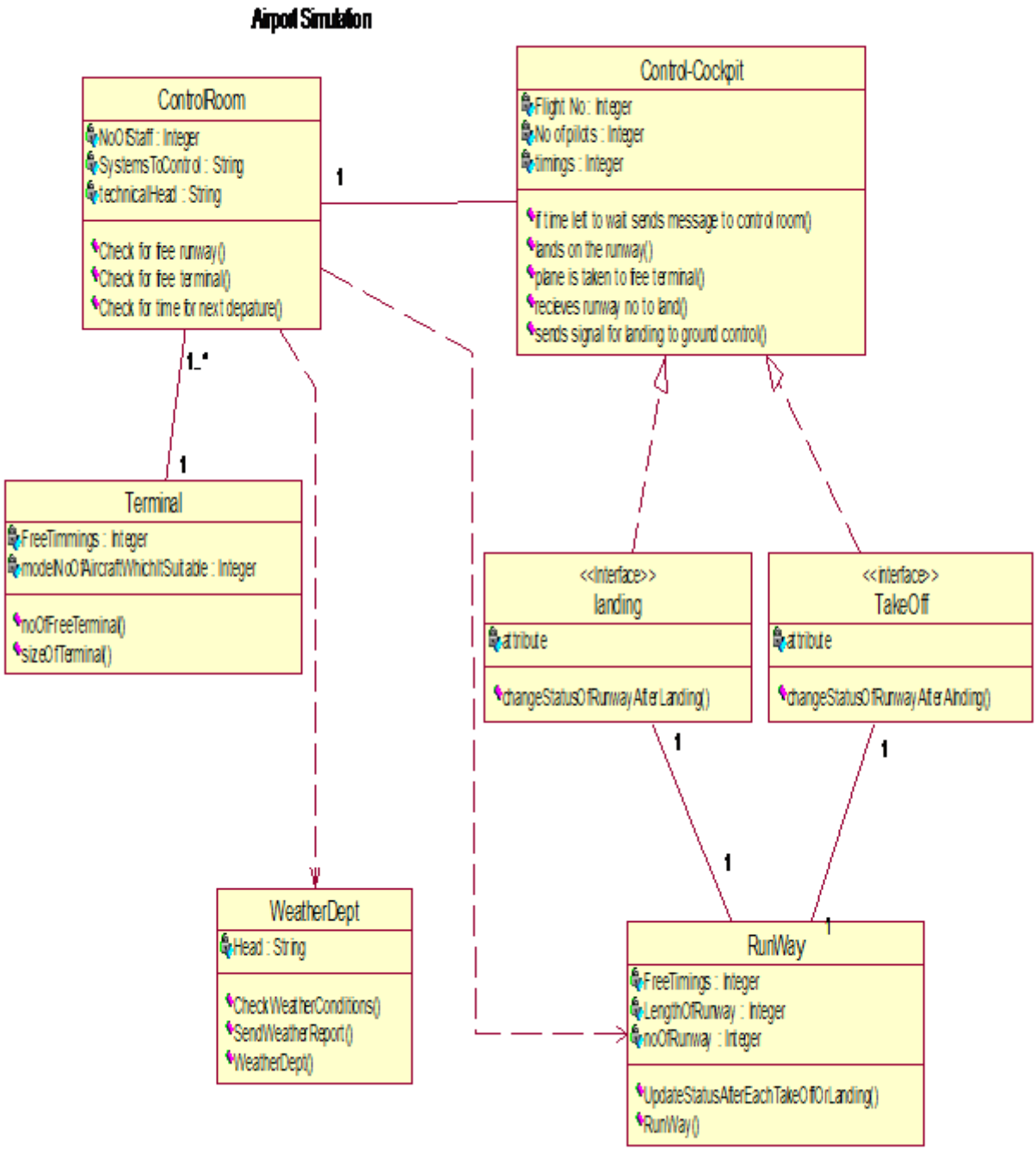
Classes:

The classes contain the attributes and operations related to them the main classes classified in this solution are:

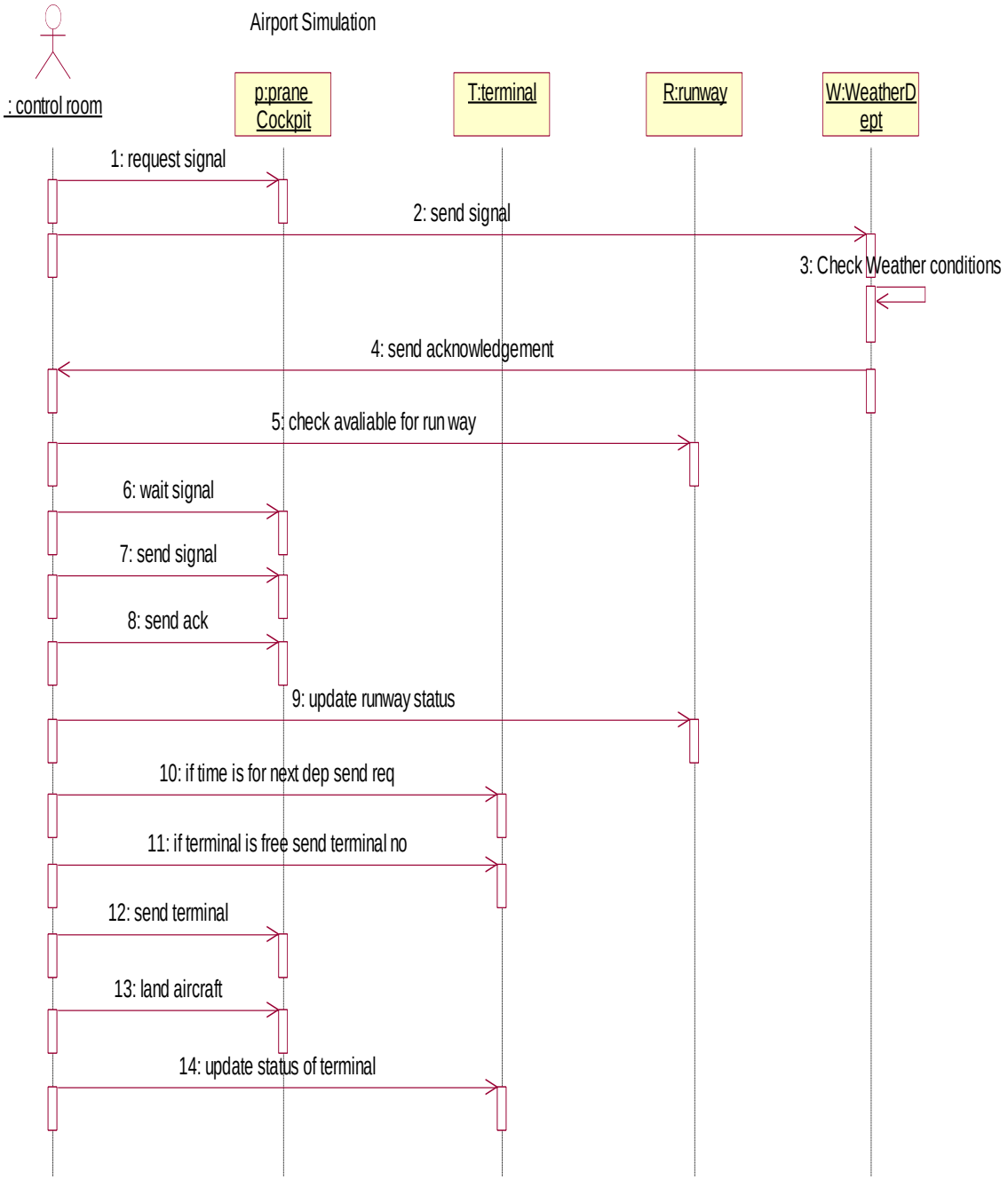
1. **Control Room:** he is the person who supervises the controls the ground traffic on runway. He checks the status of runways and assigns the free runways and terminals for takeoff and landing.
2. **Plane Cockpit:** He is the person who controls the aircraft. He transmits or receives signals regarding the free runways and terminals from the control room. He is responsible for the safe landing or takeoff of the plane.
3. **Runway:** This is the part the planes use to land or takeoff only one plane can use runway at a time to takeoff or land.
4. **Terminal:** This is the place where the planes are parked until the next departure. The terminal is to be parked in it.
5. **Takeoff/land:** The leaving of planes is called takeoff and coming back to runway is called landing. The runway is used for either purpose.

1. Use Case Diagram:

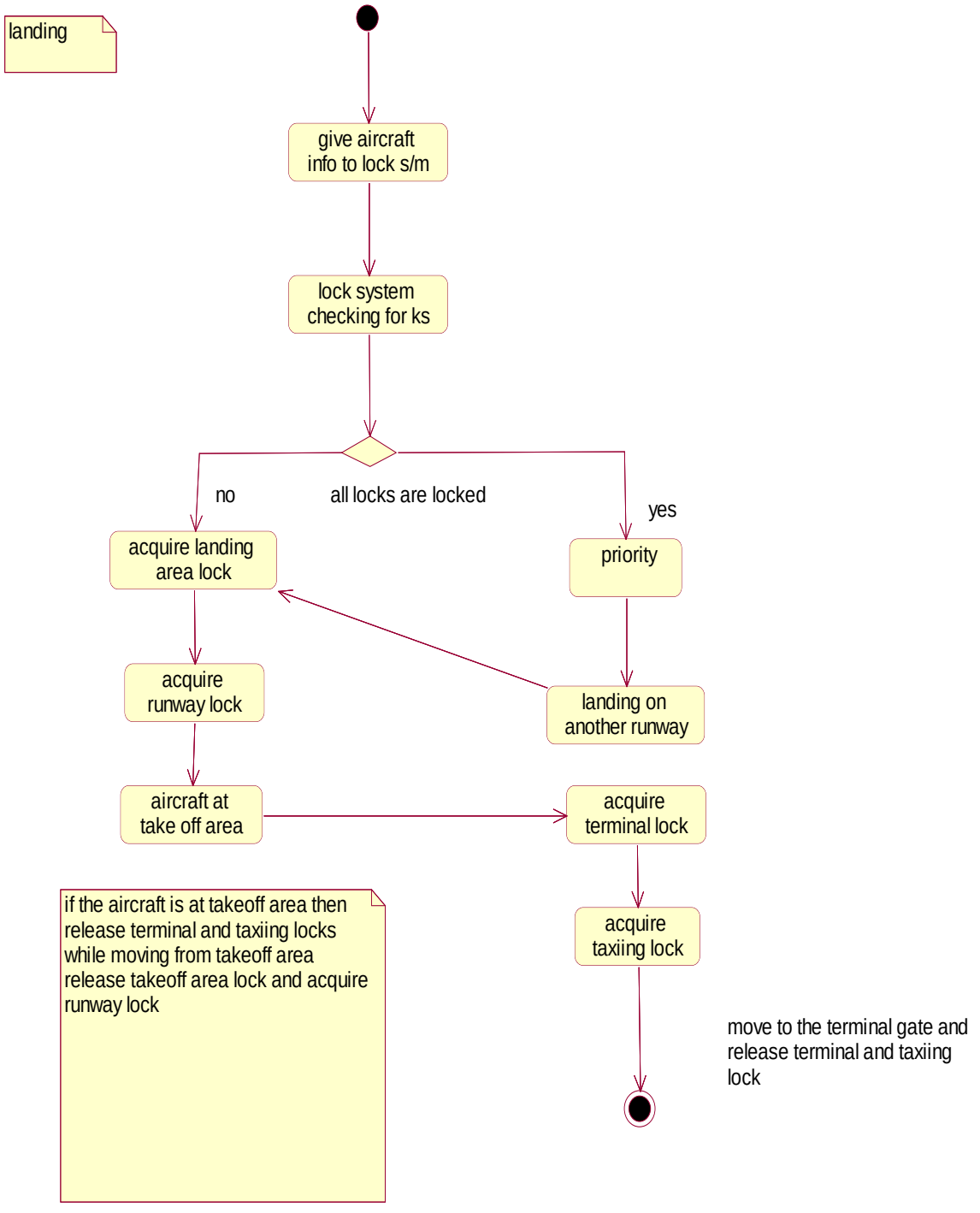
2. Class Diagram



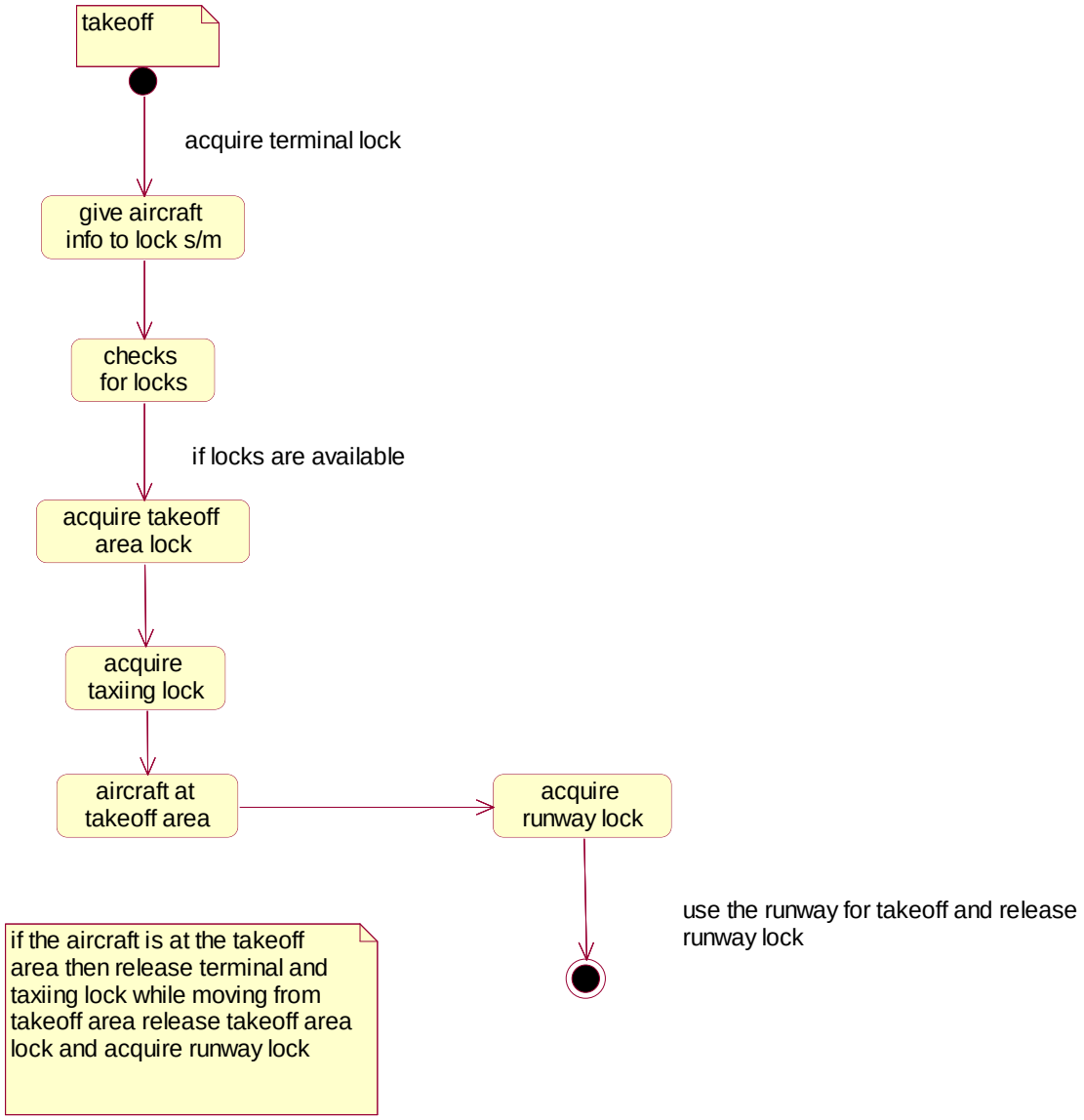
3. Sequence Diagram:



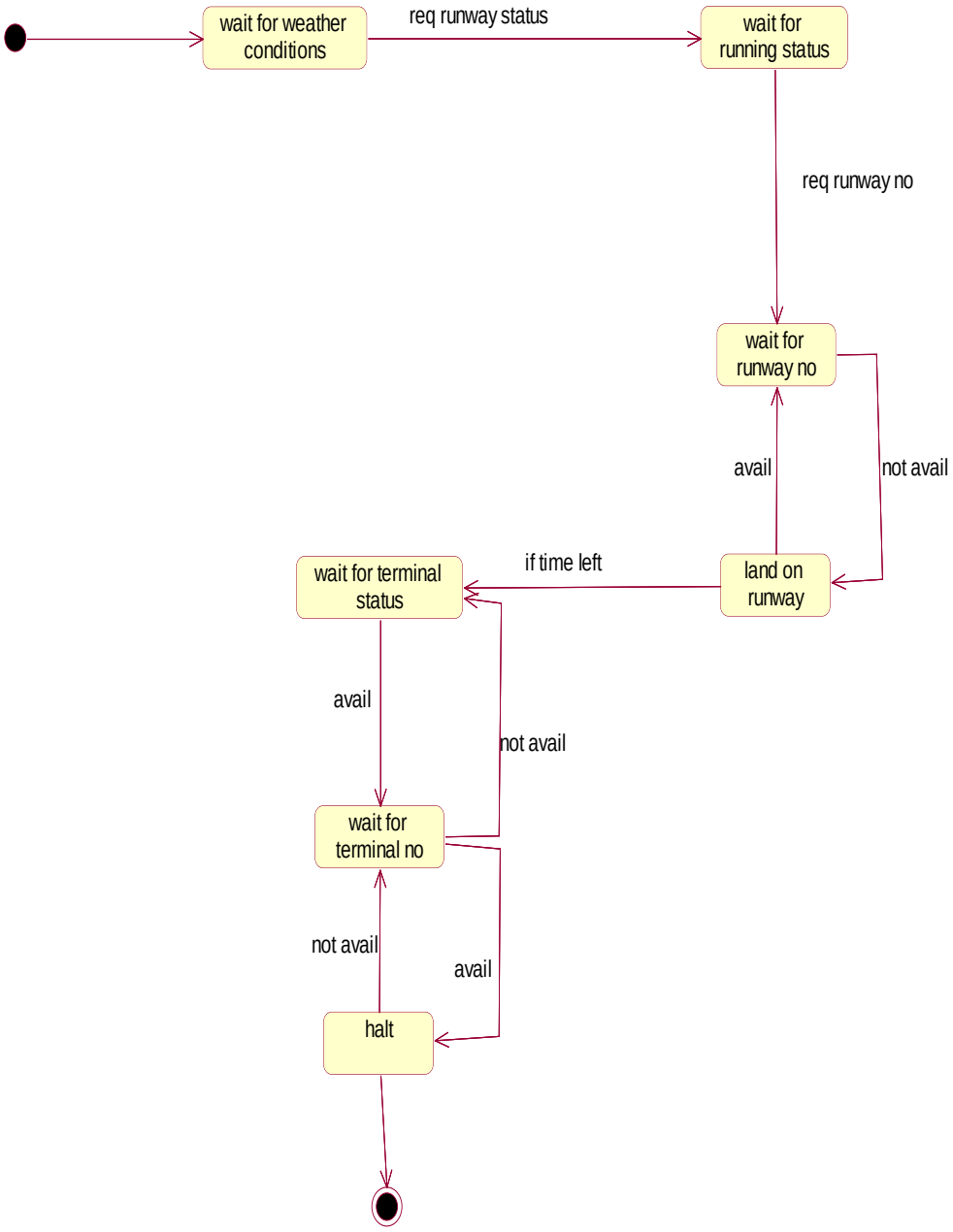
4. Activity Diagram:



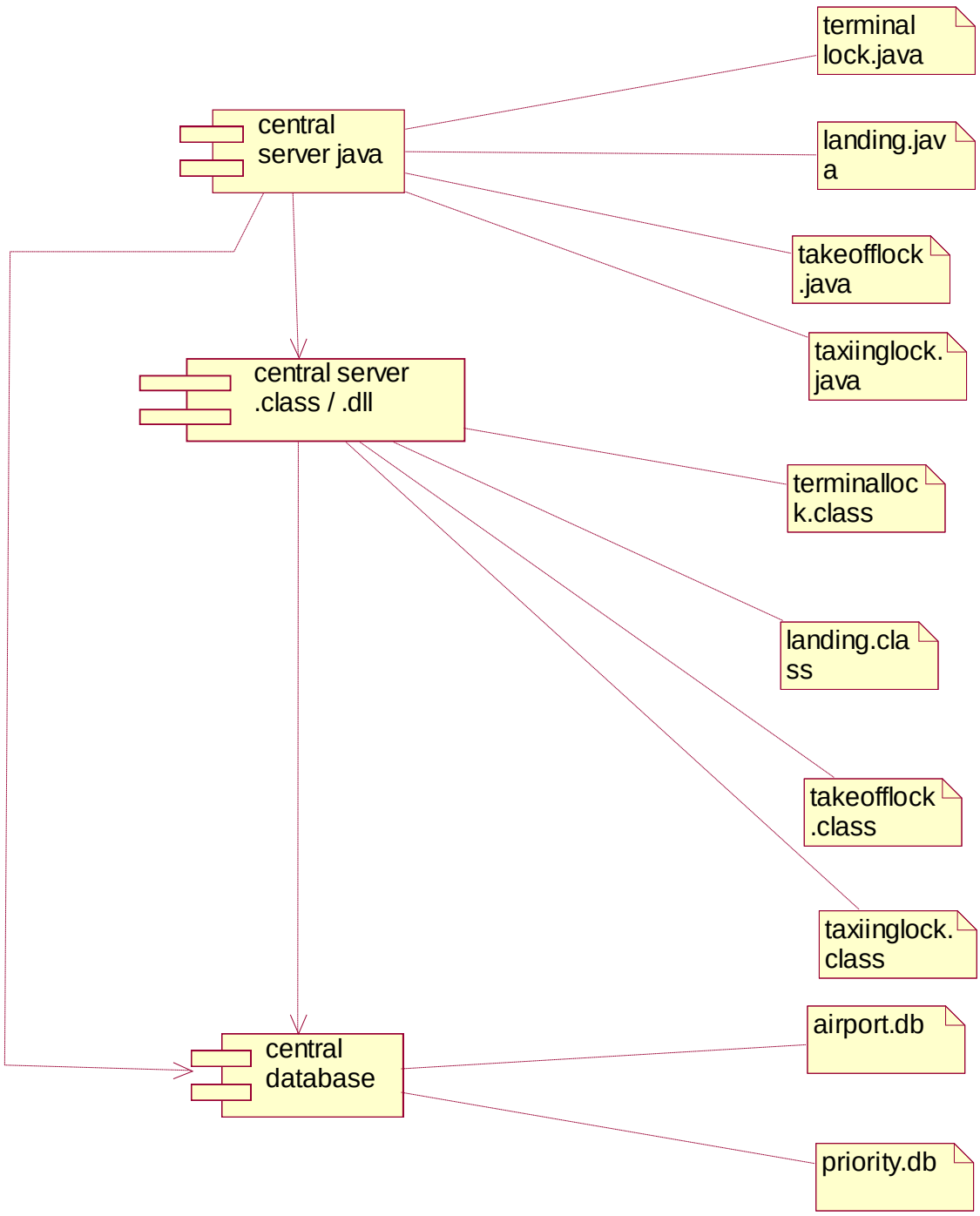
4.1 Activity Diagram2:



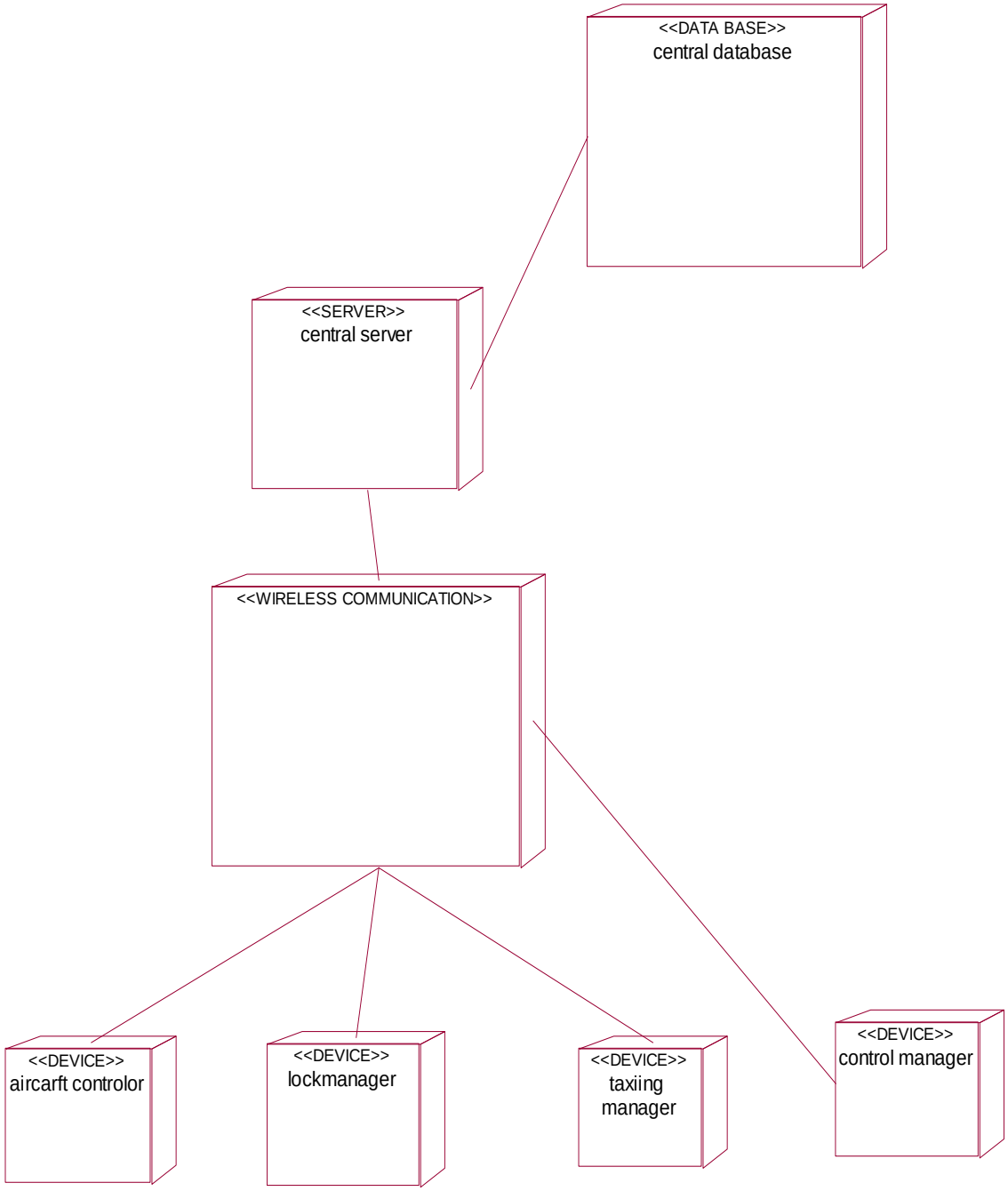
5. **State Chart Diagram:**



6. Component Diagram:



7. Deployment Diagram:



Project-V: An Automated Community Portal: Business in the 21st Century is above all BUSY. Distractions are everywhere. The current crop of "enterprise intranet portals" is often high noise and low value, despite the large capital expenditures it takes to stand them up. Email takes up 30 - 70% of an employee's time. Chat and Instant Messaging are either in the enterprise or just around the corner. Meanwhile, management is tasked with unforeseen and unfunded leadership and change-agent roles as well as leadership development and succession management. What is needed is a simplified, repeatable process that enhances communications within an enterprise, while allowing management and peers to self-select future leaders and easily recognize high performance team members in a dynamic way. Additionally, the system should function as a general-purpose content management, business intelligence and peer-review application. Glasscode's goal is to build that system. The software is released under a proprietary license, and will have the following features: Remote, unattended moderation of discussions However, it will have powerful discovery and business intelligence features, and be infinitely extendable, owing to a powerful API and adherence to Java platform standards. Encourages peer review and indicates for management potential leaders, strong team players and reinforces enterprise and team goals seamlessly and with zero administration.

1. Actors - There four actors who will use "Online Job Seeker". They are

- Administrator: He is the one to maintain the website of the company.
- Moderator: His role is to add and remove the job seekers, approve and sort the vacancies in the company.
- Employer: Role is to get check their registration details of the jobseeker, view the application and CV, if any vacancies available they must add it.
- Job Seeker: view the advertisement, search and apply the vacancies in the companies, upload the CV, register in that company website for apply.

2. Functional Requirements

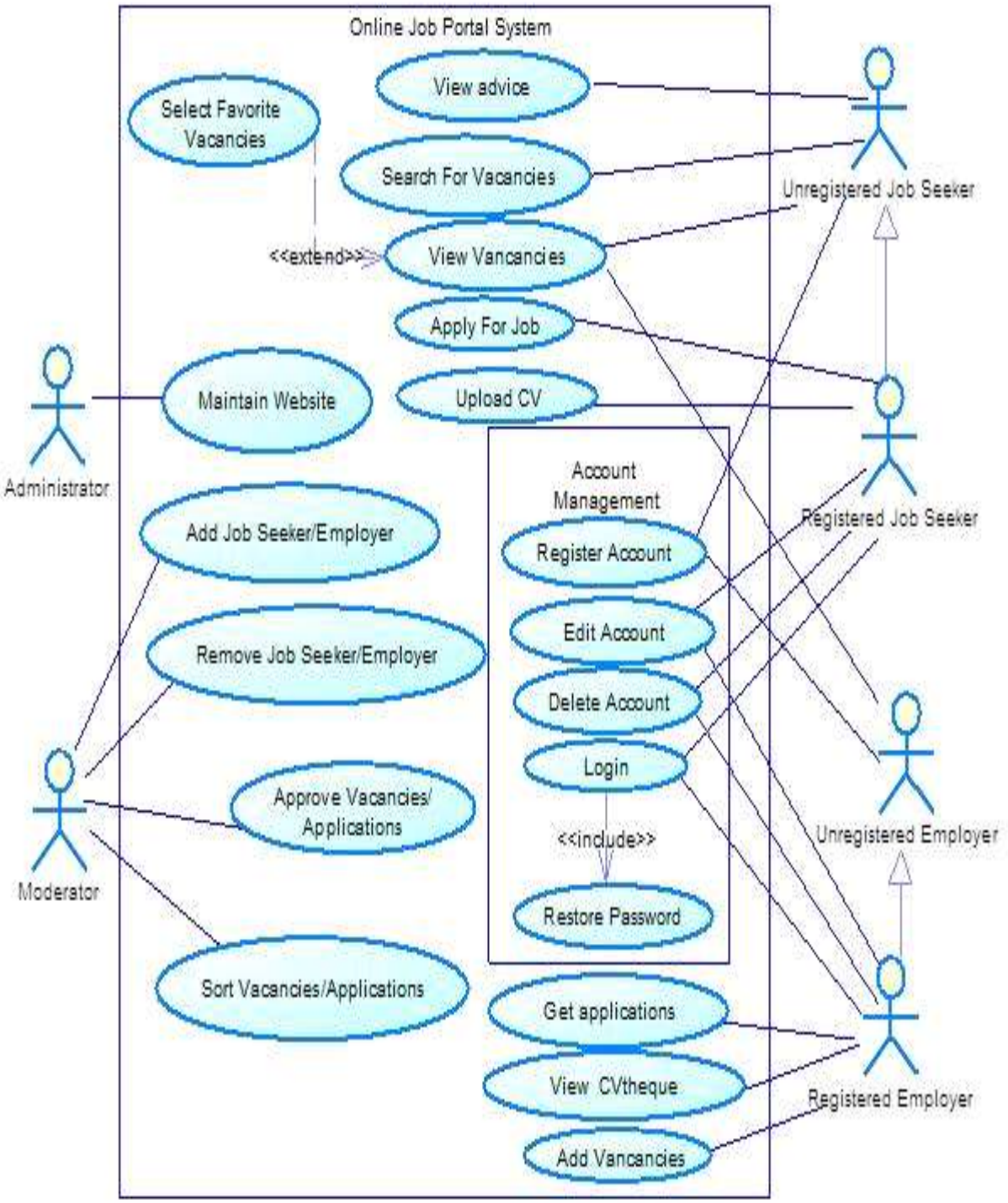
- The system supports Job Seekers Application.
- System can search the job from the company according to Job seekers demand.
- System can add job vacancies in form of advertisement.

- System can update employer's details and vacancies list in the company.
- System can delete job vacancies.
- System can show the website.
- System can register new job seeker.
- System can check the sign in of the registered job seeker and employee (user ID and Password)
- System can get applications from job seeker.
- System can allow to upload CV and application form.
- System can view all the service records according to Employee ID.
- System can update password (Admin & Employee).

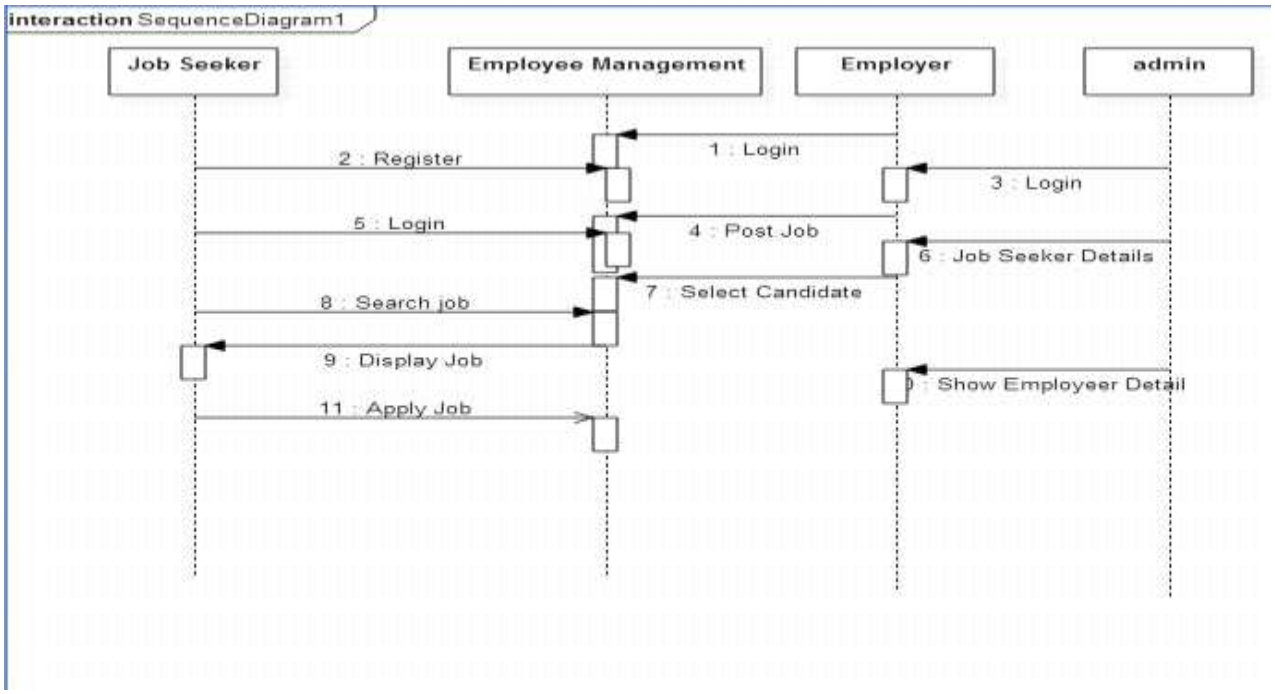
3. Non-Functional Requirements

- The system can save employee details into the database safely.
- The system can support all the PC (Personal Computer).
- The system can create a backup database file after every employee joined in the company (authentication details).
- For security issues only, admin can change the password on behalf of Employee.
- The job portal will provide restriction against unauthorized access.
- There will be a backup of data for any future mishap
- The online job portal will never break down and work consistently.

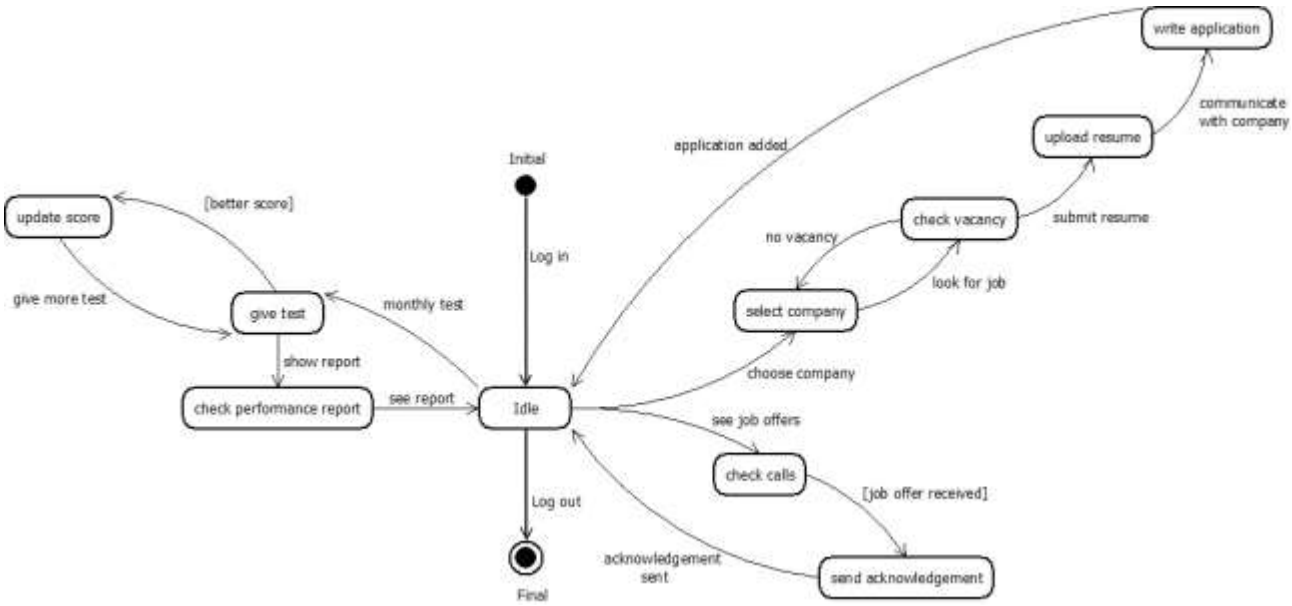
4. Use Case Diagram



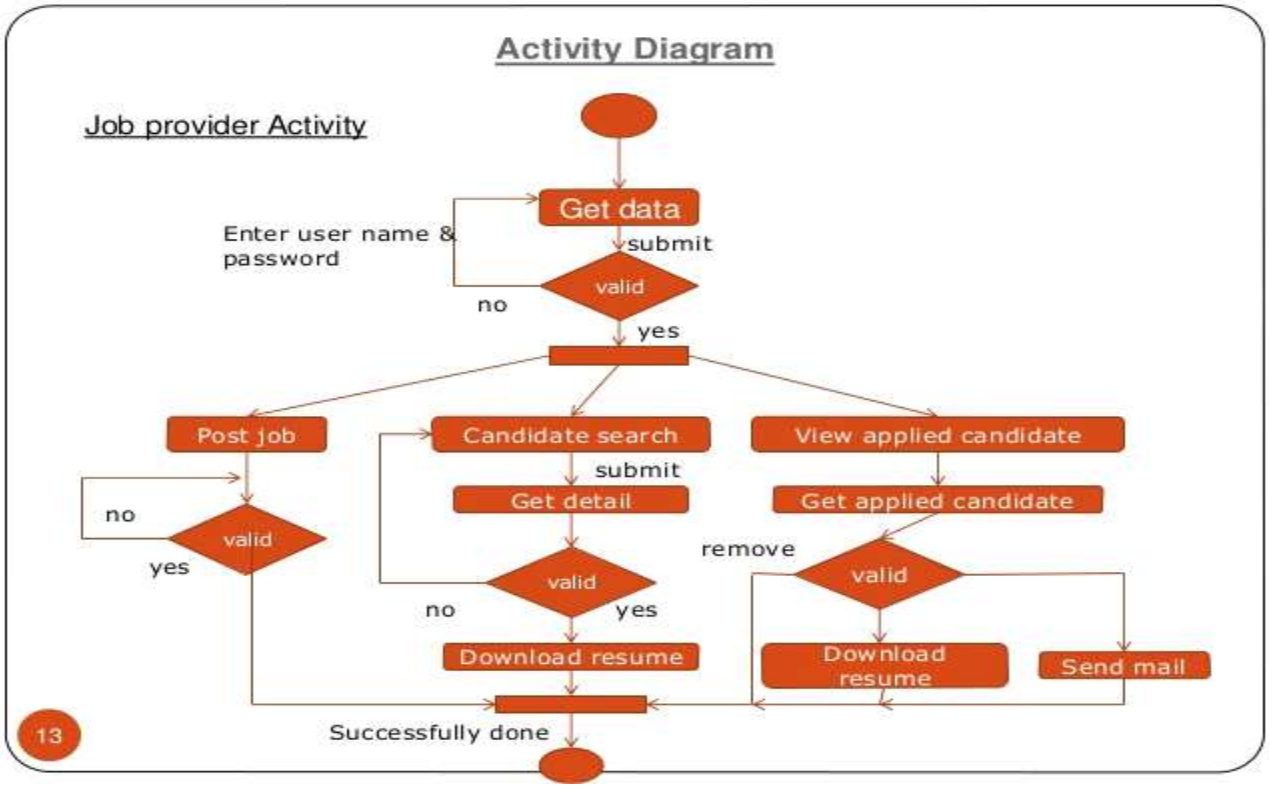
5. Sequence Diagram

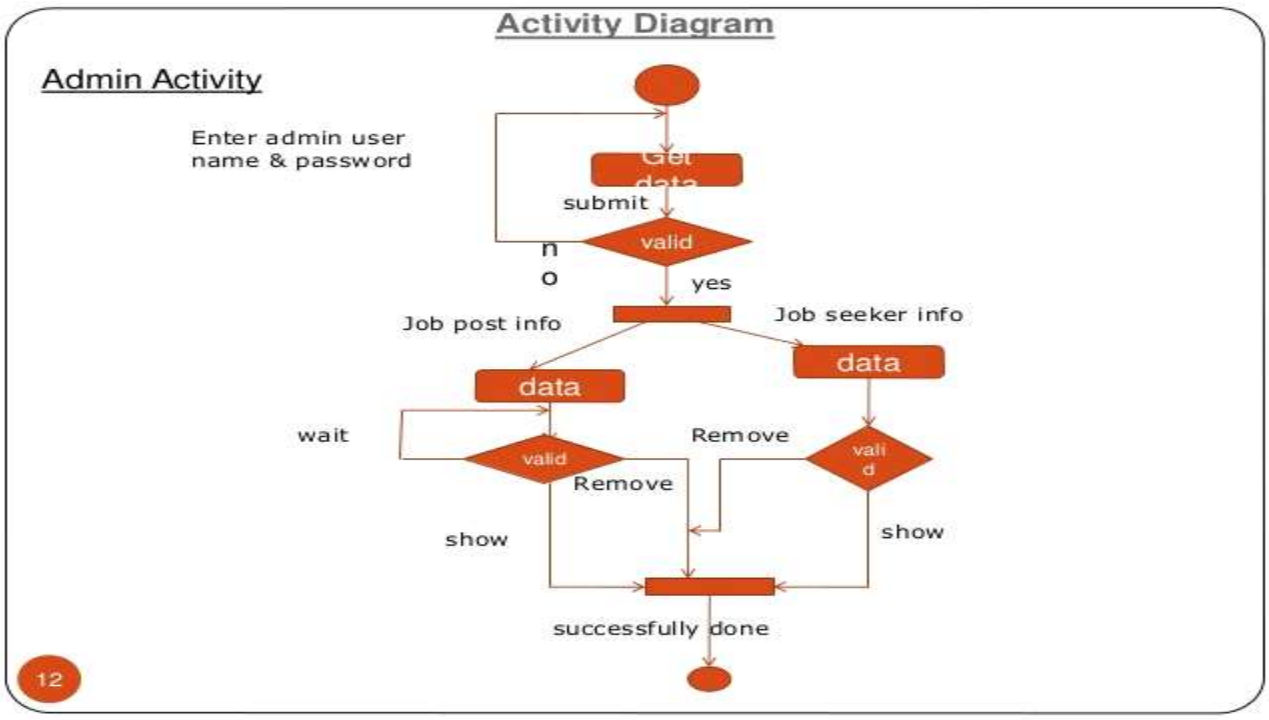


6. State Chart Diagram

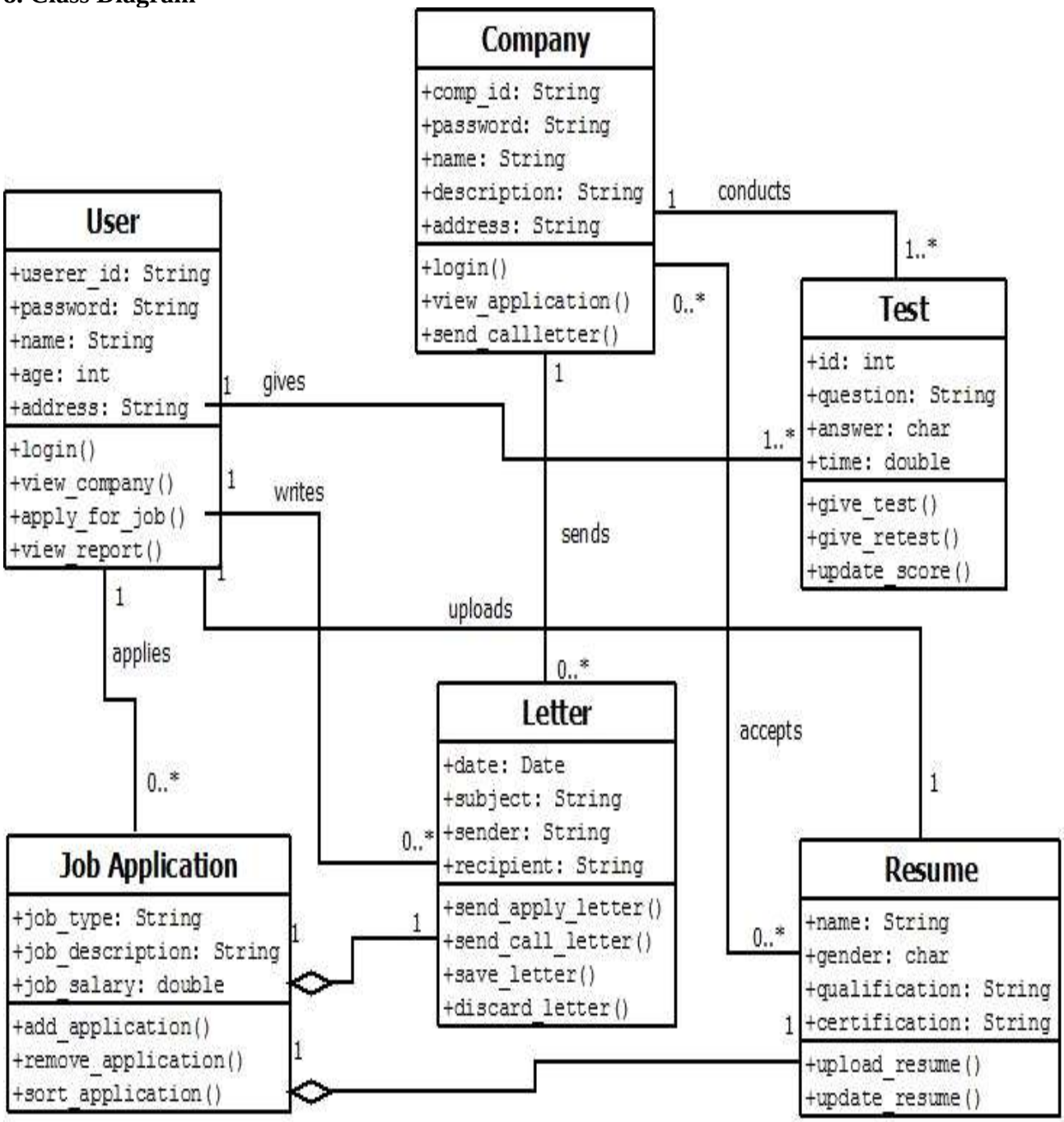


7. Activity Diagram

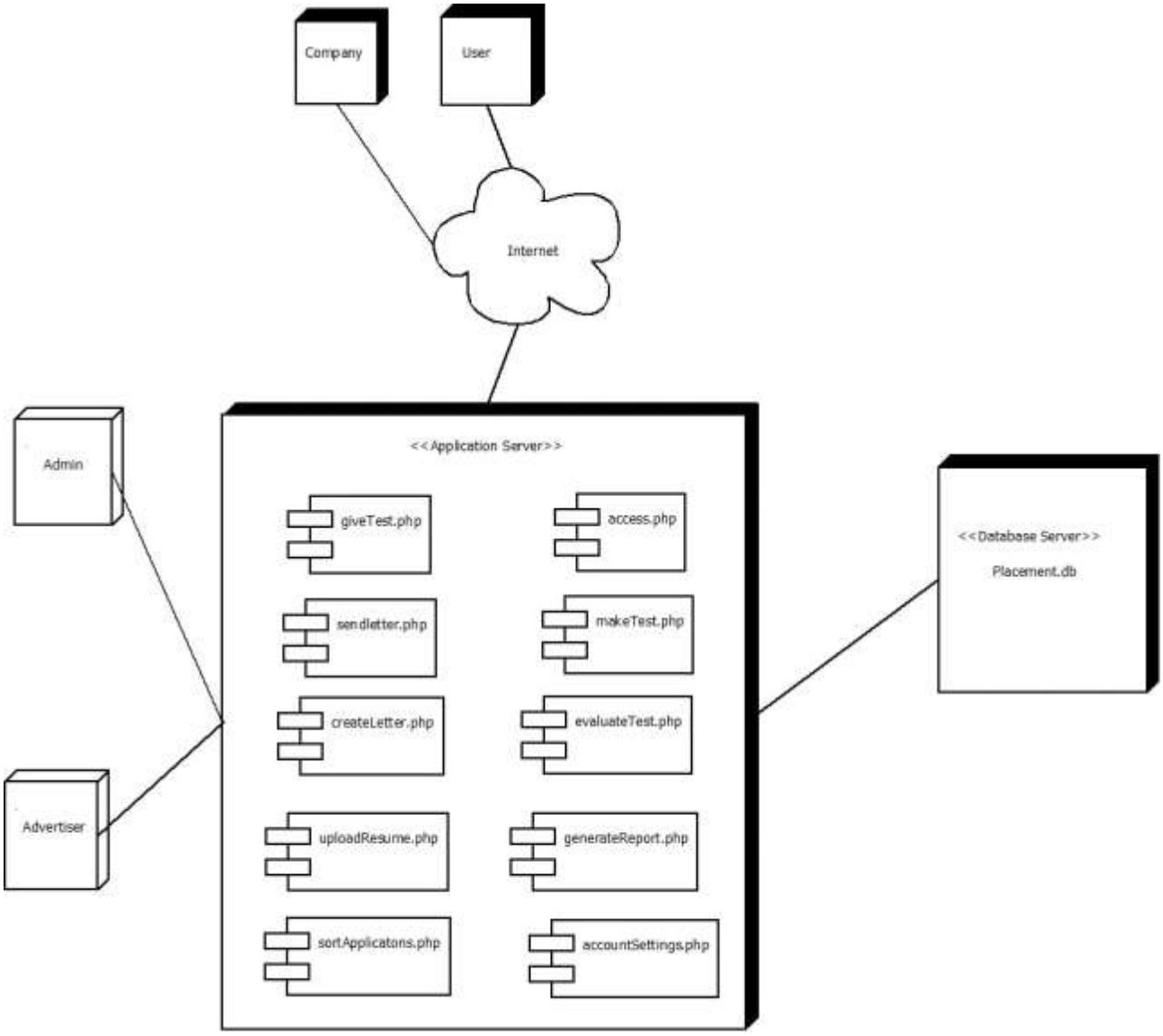




8. Class Diagram



9. Component Diagram



Project-VI: Content Management System: A content Management system is a system used to manage the content of the website. The CMS will be maintained by the content manager or author to manage the creation, modification, and removal of content from a website. It allows a large number of people to contribute and to share the stored data. CMS will control access to data, based on user roles. User roles define information each user can view or edit or share. It aids in easy storage and retrieval of data. A CMS indexes all data within an organization. Individuals can then search for data using keywords, which the CMS retrieves.

The main goal is to enable non-technical end users to easily publish, access, and share information over the web, while giving administrators and managers complete control over the presentation, style, security, and permissions

Features:

- Robust Permissions System
- Templates for easy custom site designs
- Total control over the content
- Search engine friendly URL's
- Role based publishing system
- Versioning control
- Visitor profiling

1. **Content Management System in detail: A content Management System** is a collection of procedures used to manage work flow in a collaborative environment. E availability of content search has revolutionized the way people discover information, yet search maintain larger and larger indexes. CMS takes the keywords from the user of their information need, and delivers the result from the indexing server of their organization.

   In order to access the data, initially user has to register in to the form by filling all the fields in the registration form. Then the user will enter into the website for getting login to the web form. By validating the details of the user, he will enter to the site.

   User will be able to browse the content after login. He can view the content which includes of user can able to delete the content which he has not required, user can able to copy the content, edit the content of his own, and he can able to share the content to the other users of his own.

Coming to the admin role, the main aim of the admin is to, he should able to maintain all the website online content. For this, admin should also get login to his account by checking his credentials. Maintenance of online content includes that admin can able to delete the content, update the content of his own, and he can add the new content.

The aim of admin is also include to maintain the user database which includes adding new user details to the data base, deleting the old users or users which he don't want and he can able to update the existing user specified details.

2. **Actors:** CMS include of two actors:
   i. User
   ii. Admin/ Content developer

User: User can able to access the content and he can share/ edit the data.

Admin: Admin has to maintain all the content in the database and users details.

4. Functional Requirements :

xiii. The system supports customers searched content.
xiv. System can search the content from the server according to customers text.
xv. user can add content.
xvi. user can update content.
xvii. user can delete content.
xviii. user can edit the content.
xix. User should register to the page.
xx. User should login into the page
xxi. admin can register new staff.
xxii. admin can add new content.
xxiii. Admin has to maintain content database
xxiv. Admin maintain user details.
xxv. System can view all the service content according to user needs.
xxvi. System can update password (Admin & user).

5. Non-Functional Requirements:

xii. The system can save content into the database safely.

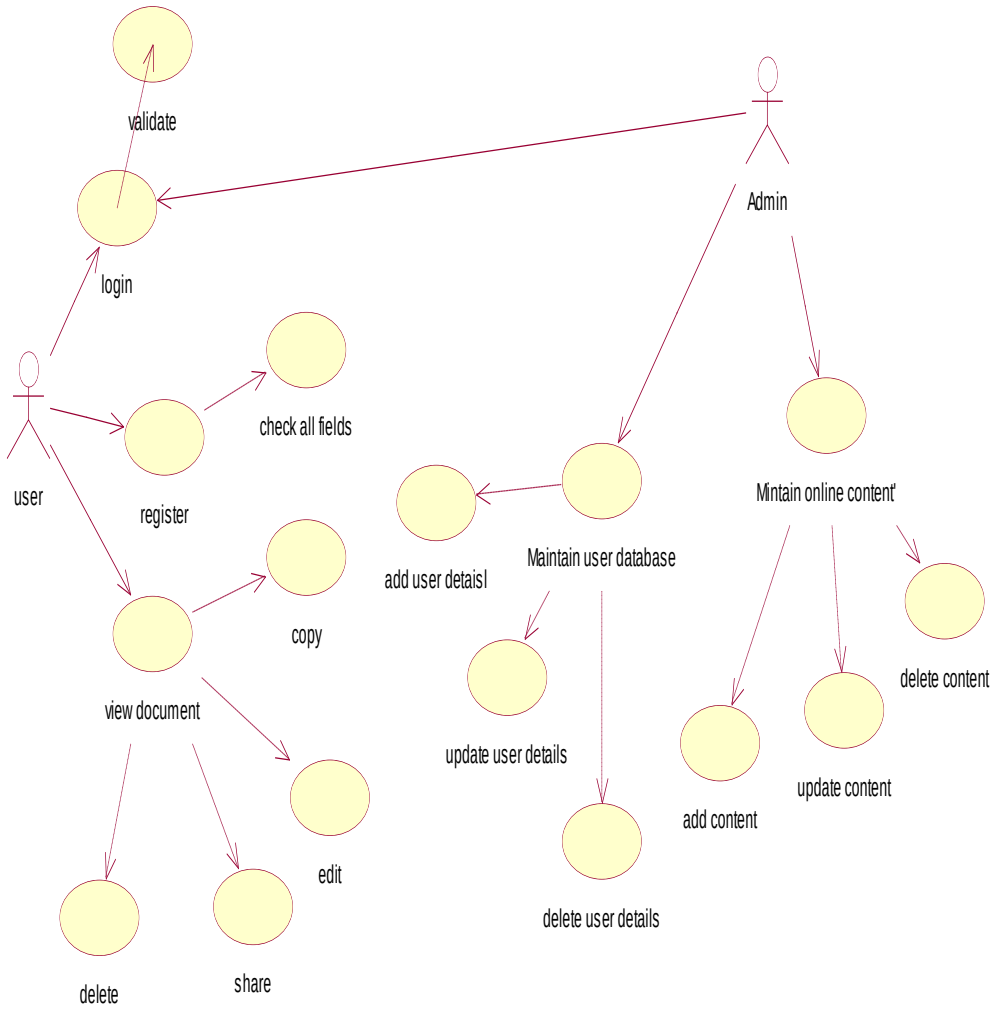
xiii. The system can support all the PC (Personal Computer).

xiv. The system can create a backup database file after every content updation and authentication details.

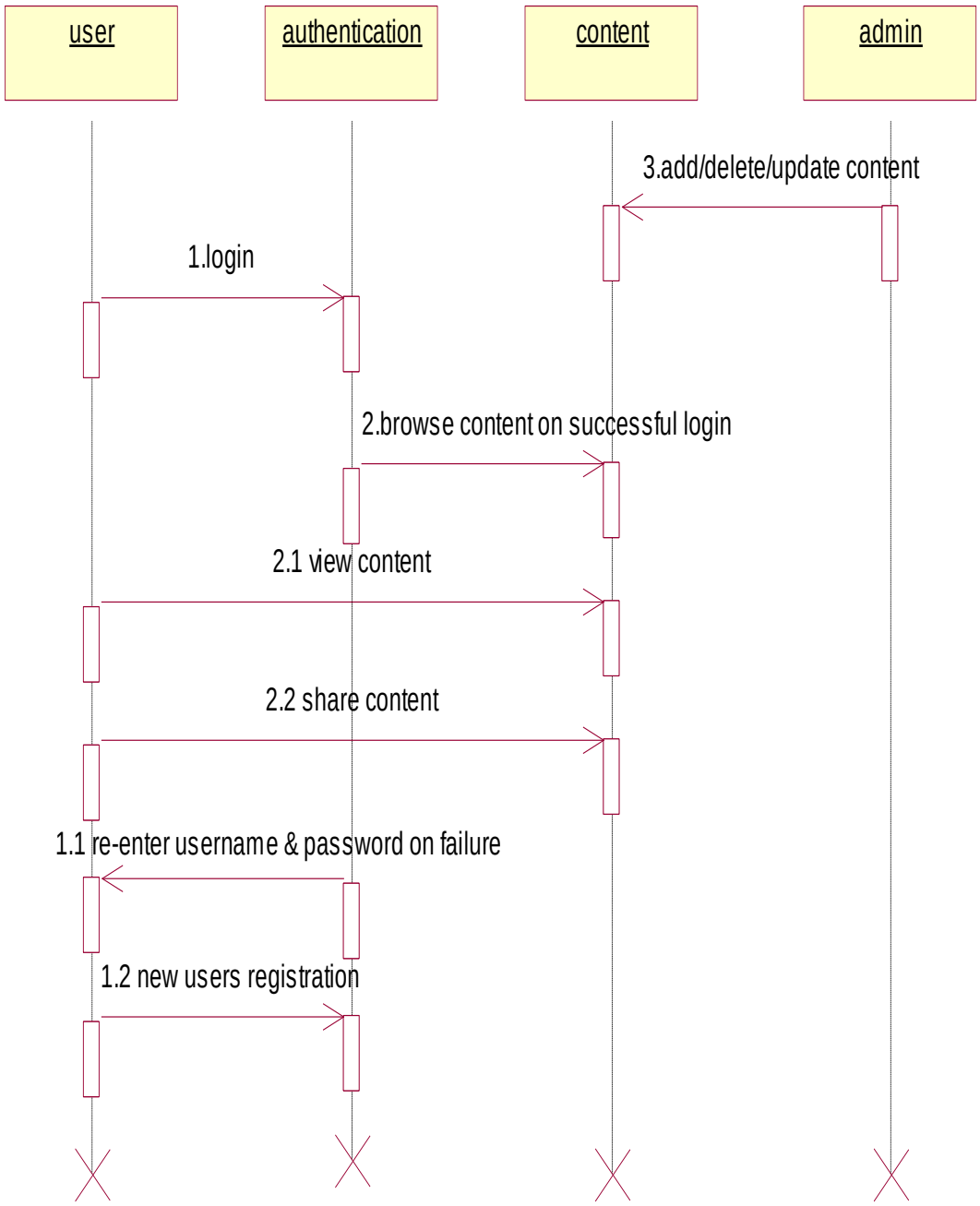
xv. For security issues only admin can change the password on behalf of staffs.

xvi. users can only access this system for content.

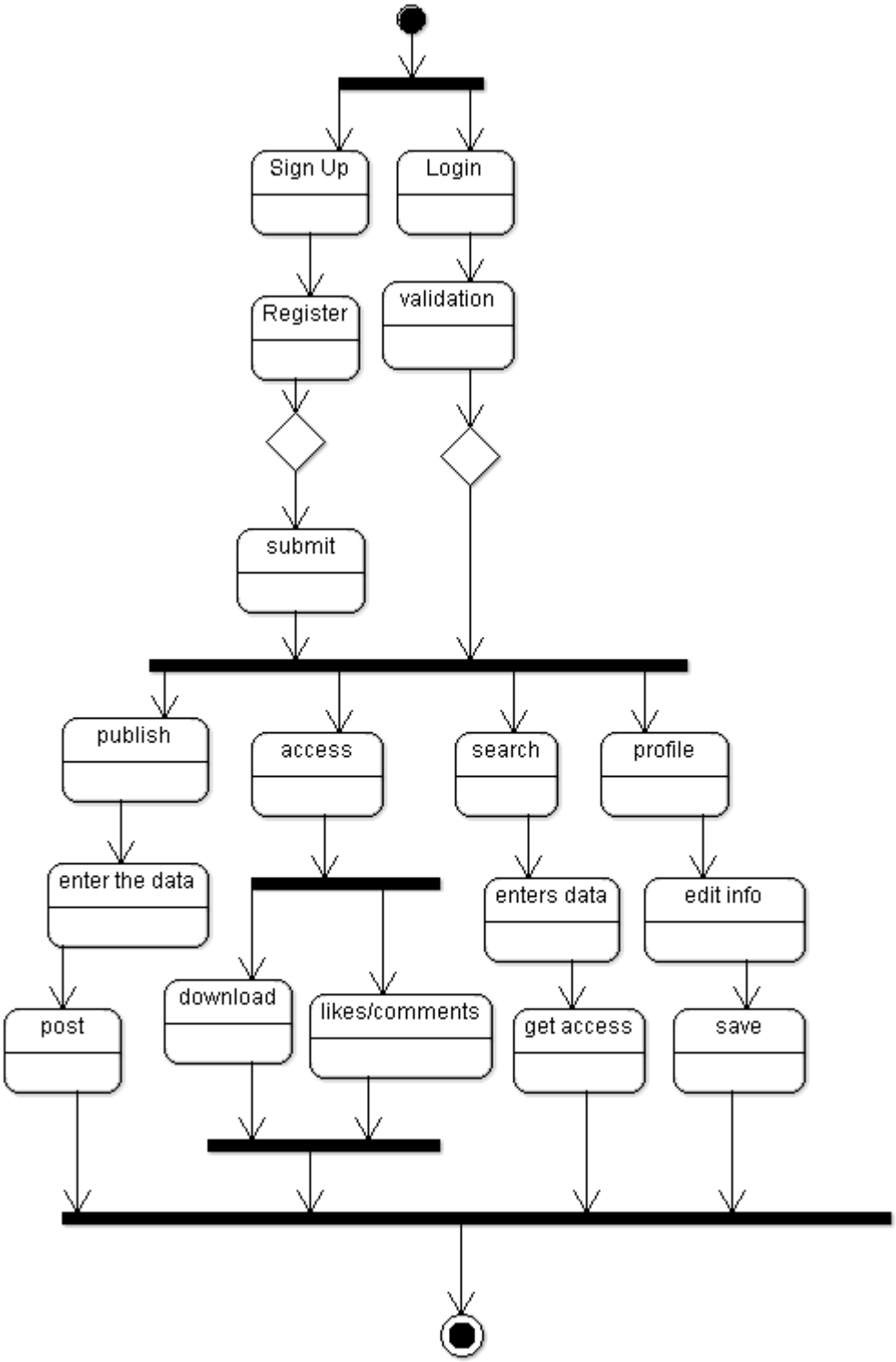
6. Use Case Diagram:



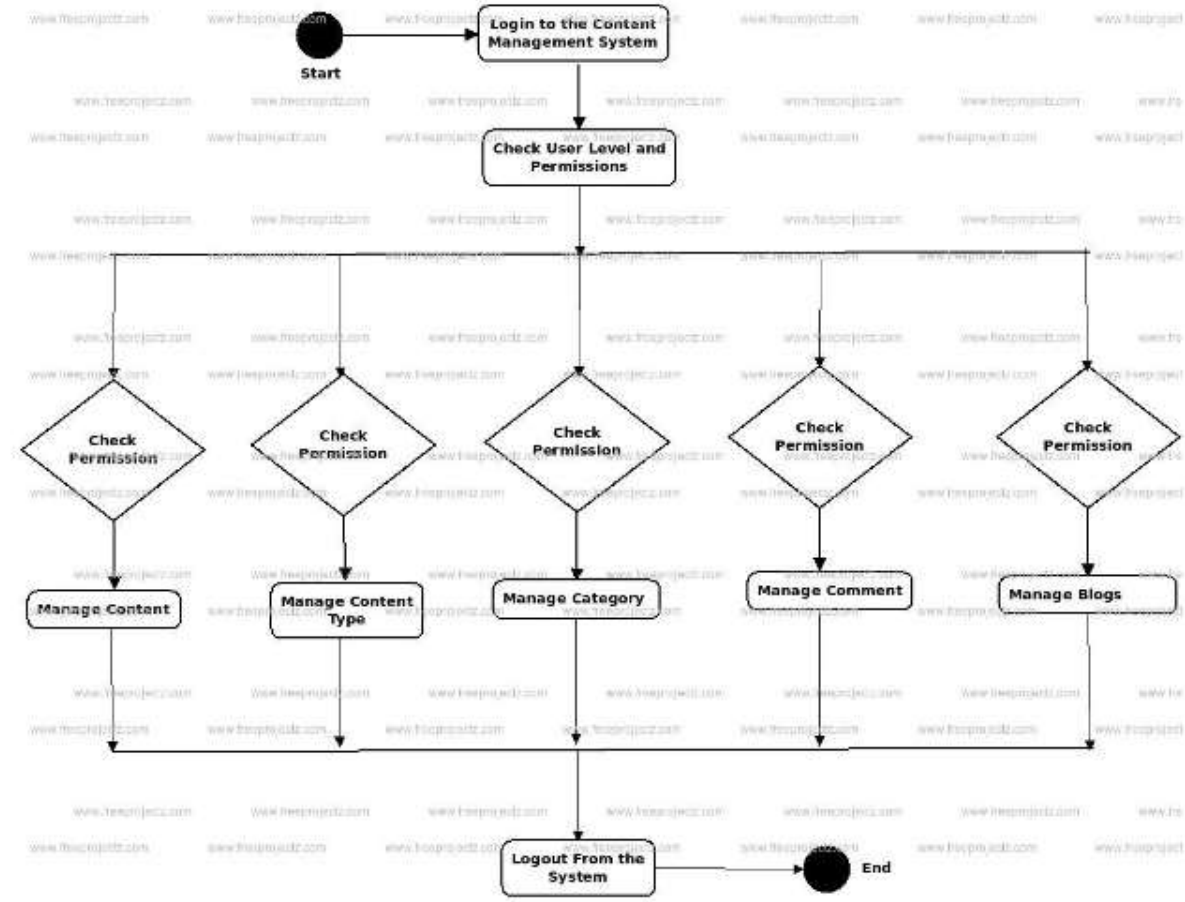
7. Sequence Diagram

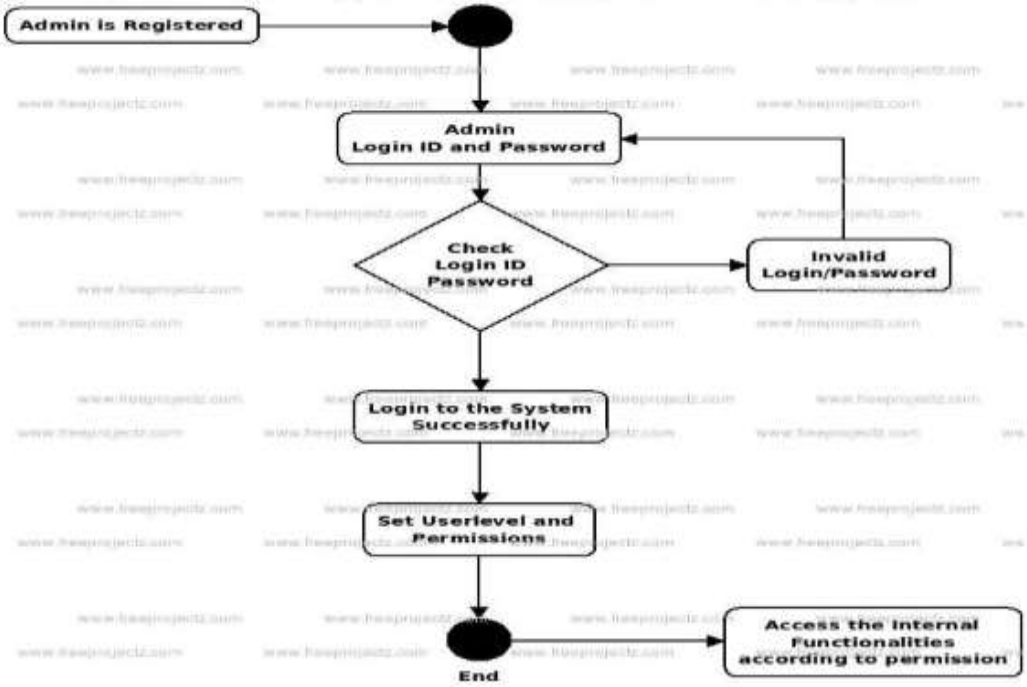


8. State Chart Diagram:

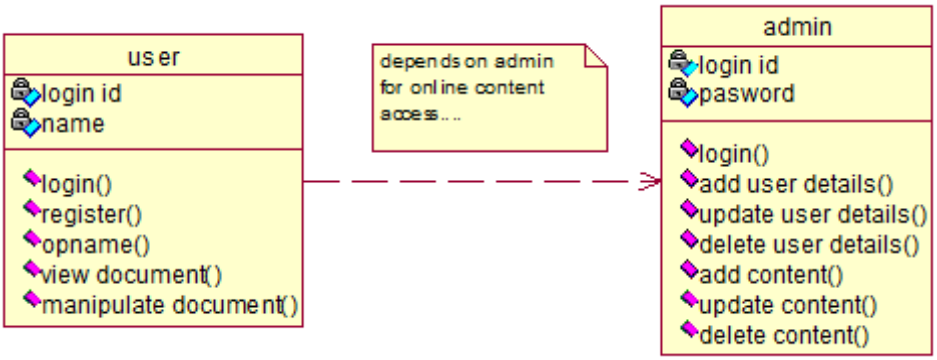


9. Activity Diagram:

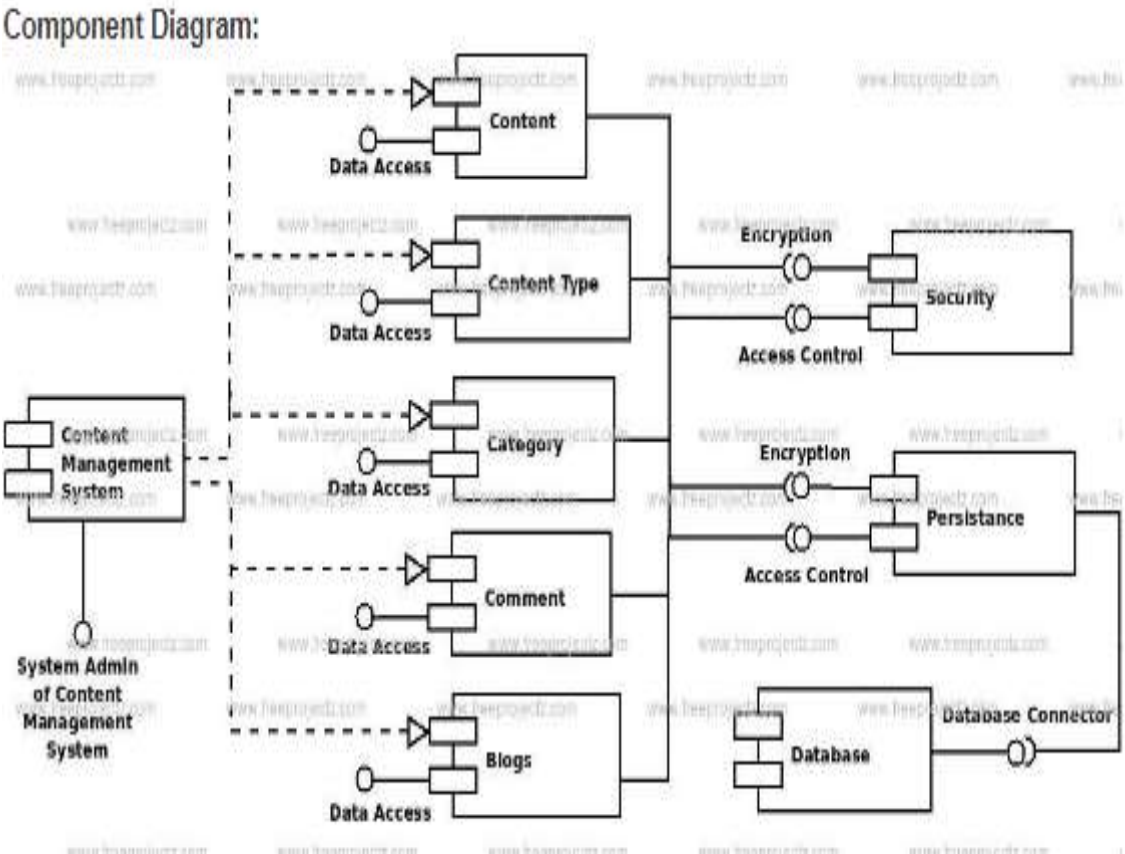




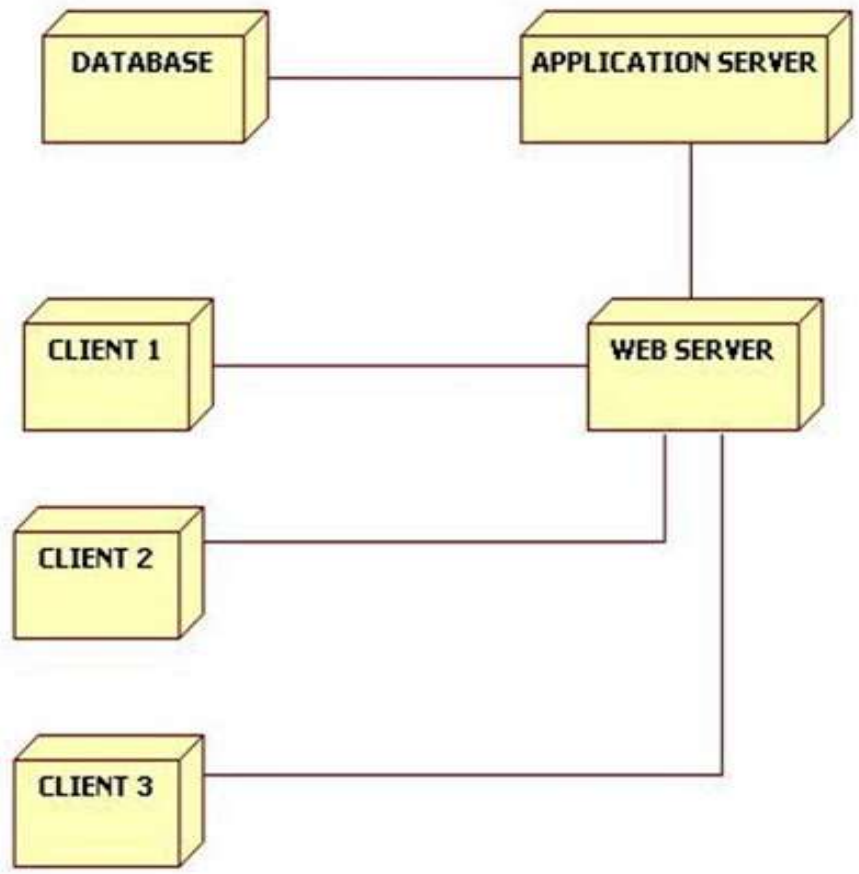
10. class diagram:



11. Component Diagram:



12. Deployment Diagram:



1. **Project-VII: An Auction Application:** Several commerce models exist and are the basis for a number of companies like eBay.com, pricellne.com etc. Design and implement an auction application that provides auctioning services. It should clearly model the various auctioneers, the bidding process, auctioning etc.
2. **An Auction Application in detail:** An online auction is also known as a virtual auction. An online auction is a service in which auction users or participants sell or bid for products or services via the Internet. Virtual auctions facilitate online activities between buyers and sellers in different locations or geographical areas. Various auction sites provide users with platforms powered by different types of auction software.
   Online auctions are a widely accepted business model for the following reasons:

   - No fixed time constraint
   - Flexible time limits
   - No geographical limitations
   - Offers highly intensive social interactions
   - Includes a large numbers of sellers and bidders, which encourages a high-volume online business

   An **auction** is a process of buying and selling goods or services by offering them up for bid, taking bids, and then selling the item to the highest bidder. The open ascending price auction is arguably the most common form of auction in use today. Participants bid openly against one another, with each subsequent bid required to be higher than the previous bid. An **auctioneer** may announce prices, bidders may call out their bids themselves or bids may be submitted electronically with the highest current bid publicly displayed.

   Online auctions include business to business (B2B), business to consumer (B2C), and consumer to consumer (C2C) auctions. Ebay is the best example of an auction site that uses all three methodologies. The online auction business model continues to evolve according to market needs. Examples include eBay, Web Store, Online Auction and Overstock. Ebay and other providers encourage legitimate bidding activity through bidder block lists. EBay also offers Dutch auctions for large inventories, where auction bidders pay according to an item's highest sale price.

3. Actors: The Actors for an Auction Application may consider as

i. User(Seller, Buyer)

ii. Administrator

User: The user is one who is using the Auction Application. The user may be Buyer or Seller.

Seller: The Seller is an user of Auction application who is selling their goods or items.

Buyer: The Buyer is an user of Auction application who is purchasing the items or goods from the seller.

Administrator: Administrator is an actor who will maintain the entire auction application like adding and removing items.

4. Functional Requirements:

i. System can show the sales report.

ii. System can add stock.

iii. System can update stock.

iv. System can delete stock.

v. System can show the stock report.

vi. If a wrong password is given three times in succession the account will be locked for some time.

vii. Buyer can pay the money by any online payment mode

viii. System can accept different bidding amount for single item

ix. Item sales for the highest bidding amount

x. Any user may view the items without buying and bidding

5. Non Functional Requirements:

i. The system can save stock into the database safely.

ii. The system can support all the PC (Personal Computer).

iii. The system can create a backup database file after every transaction (sales, stock, service, update of authentication details).

iv. Stocks can be add and remove daily based on the sale.

v. Items already sold immediately has to delete from system.

6. Modeling of Use case Diagram:

Use case diagrams are usually referred to as **behavior diagrams** used to describe a set of actions (**use cases**) that some system or systems (**subject**) should or can perform in collaboration with one or more **external users** of the system (**actors**). Each use case should provide some observable and valuable result to the actors or other stakeholders of the system.

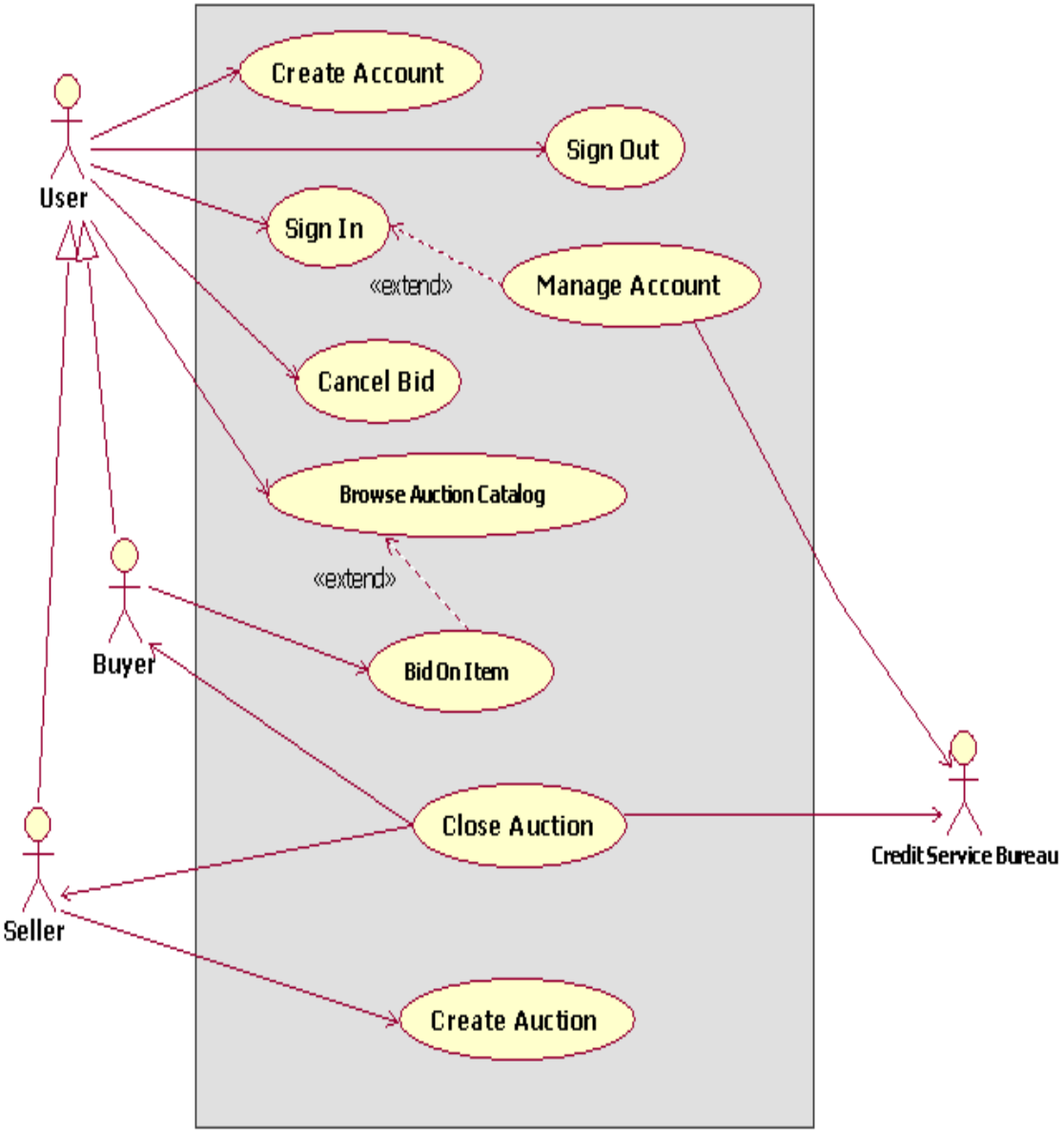


Fig: Use case diagram for Auction Application

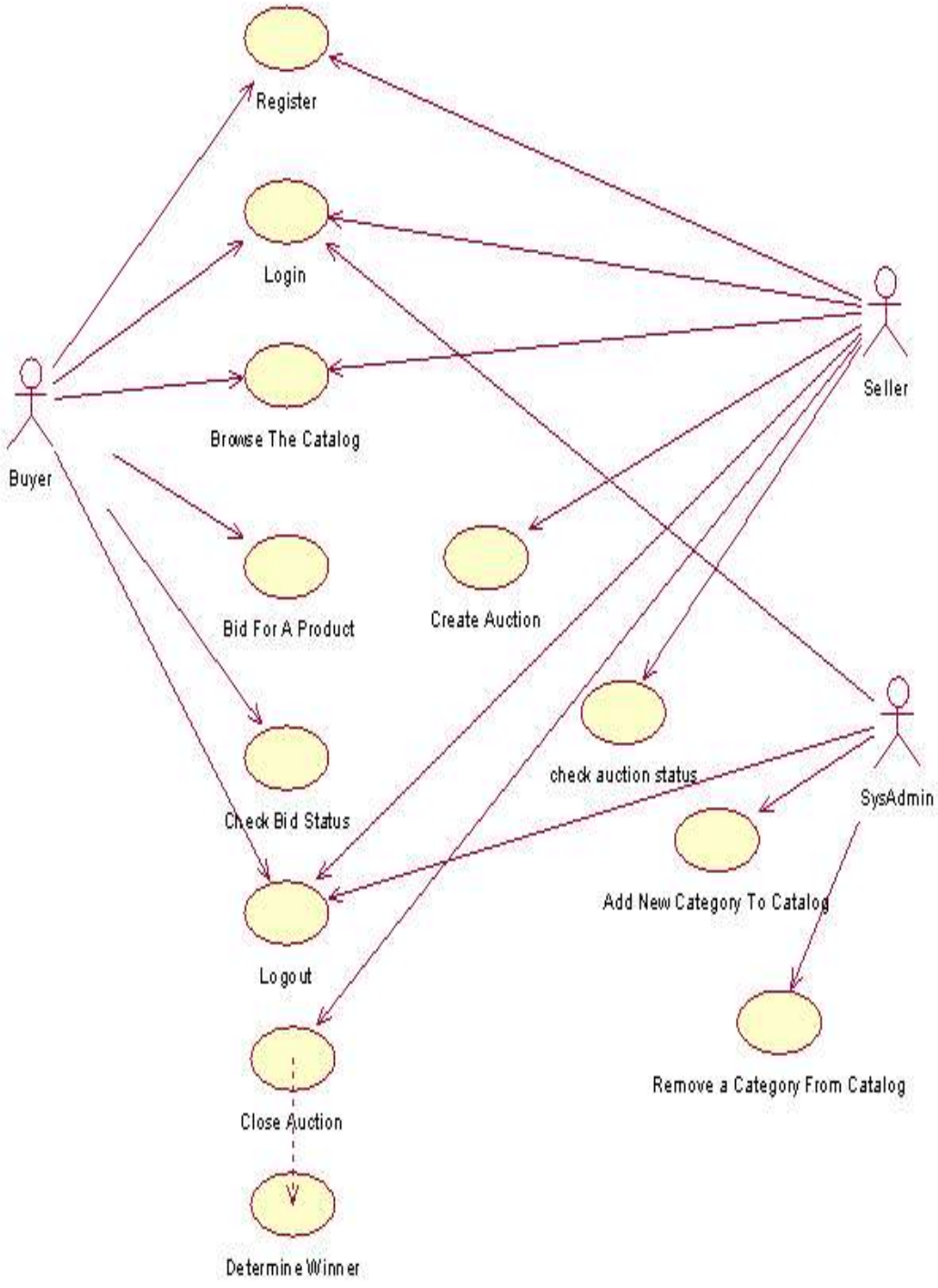
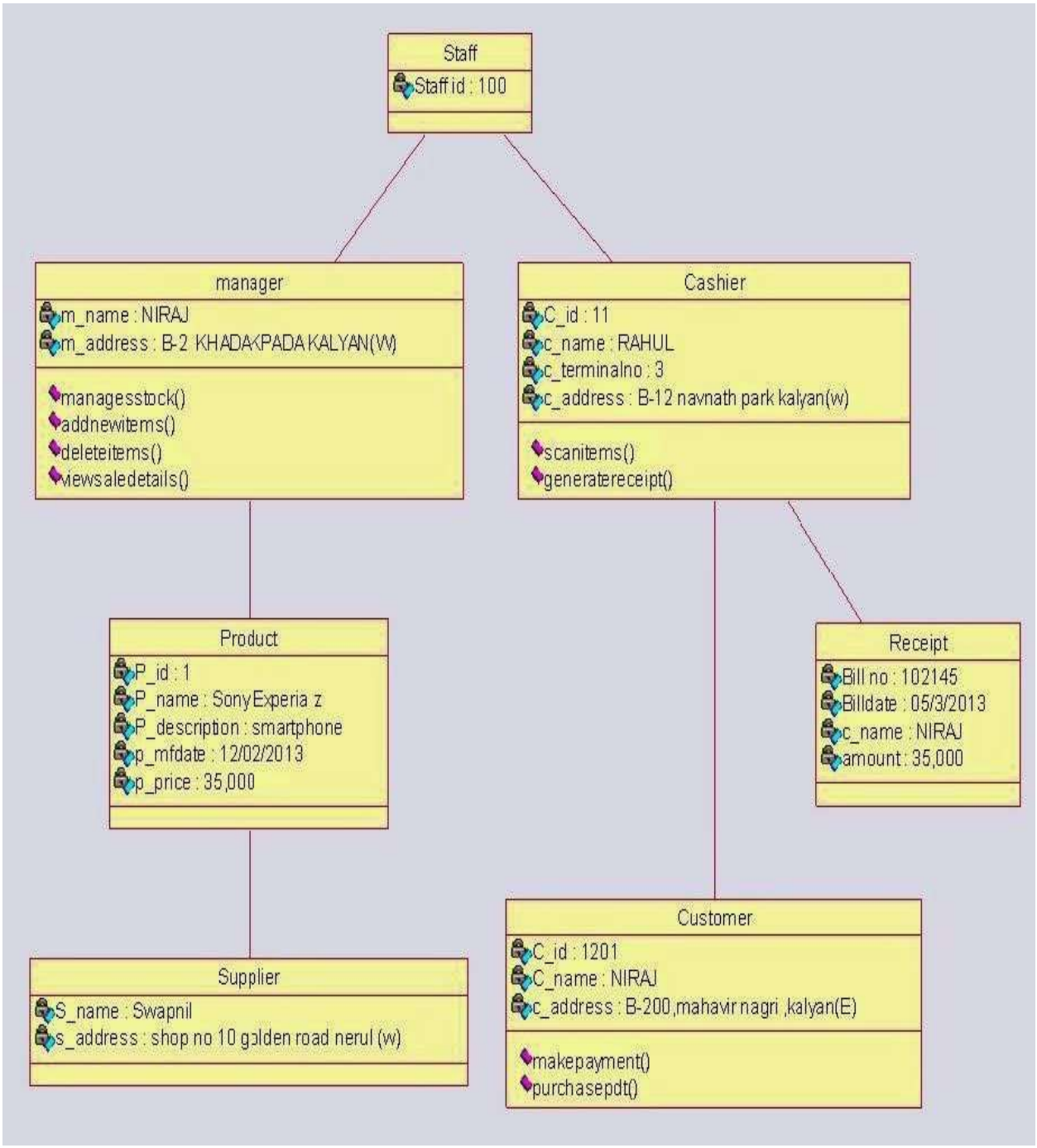


Fig: Use case Diagram for Auctio Application

7. Modeling of Class Diagram:

A **class diagram** in the Unified Modeling Language (UML) is a type of static structure diagram that describes the structure of a system by showing the system's classes, their attributes, operations (or methods), and the relationships among objects.



8. Modeling Sequence Diagram:

Sequence diagram shows object interactions arranged in time sequence. It depicts the objects and classes involved in the scenario and the sequence of messages exchanged between the objects needed to carry out the functionality of the scenario. Sequence diagrams are typically associated with use case realizations in the Logical View of the system under development. Sequence diagrams are sometimes called **event diagrams** or **event scenarios**.

A sequence diagram shows, as parallel vertical lines (*lifelines*), different processes or objects that live simultaneously, and, as horizontal arrows, the messages exchanged between them, in the order in which they occur. This allows the specification of simple runtime scenarios in a graphical manner.

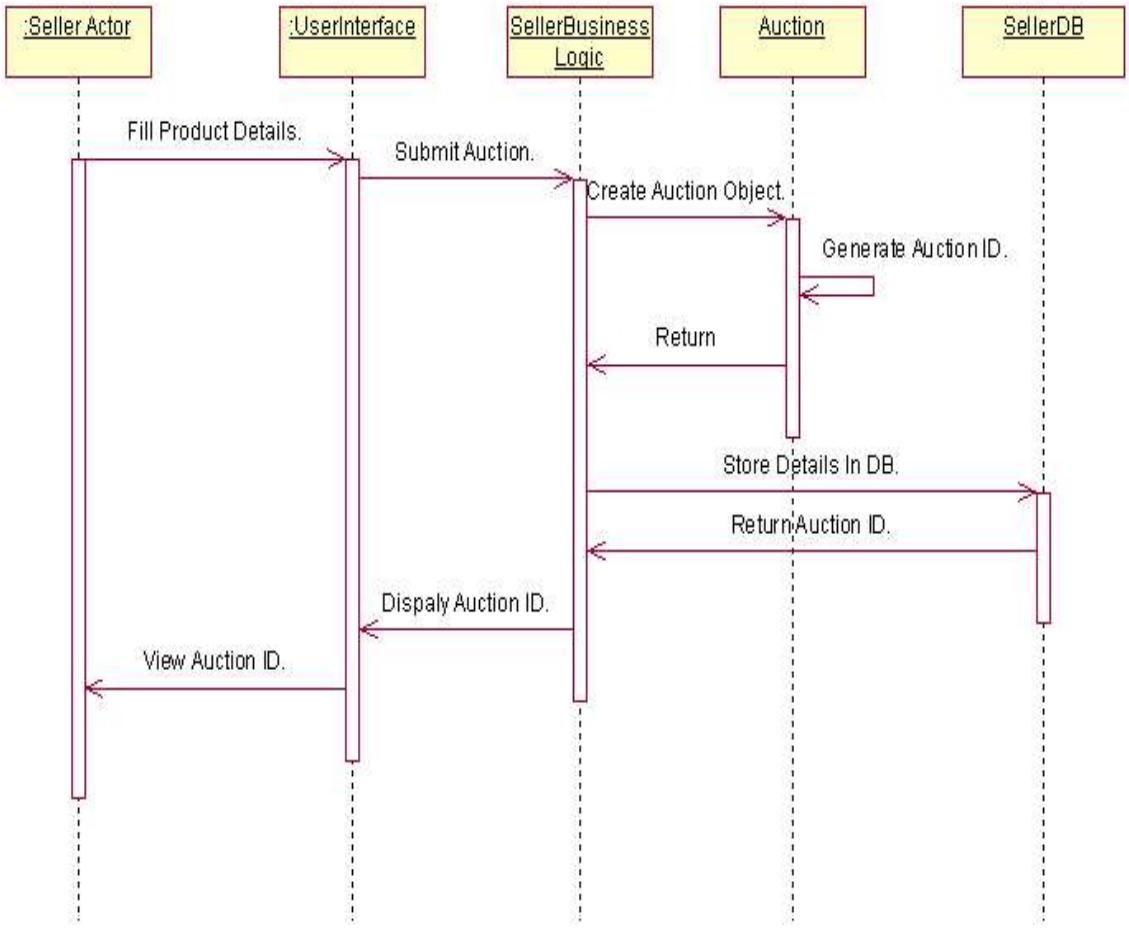


Fig: Sequence Diagram for Creation of an Auction

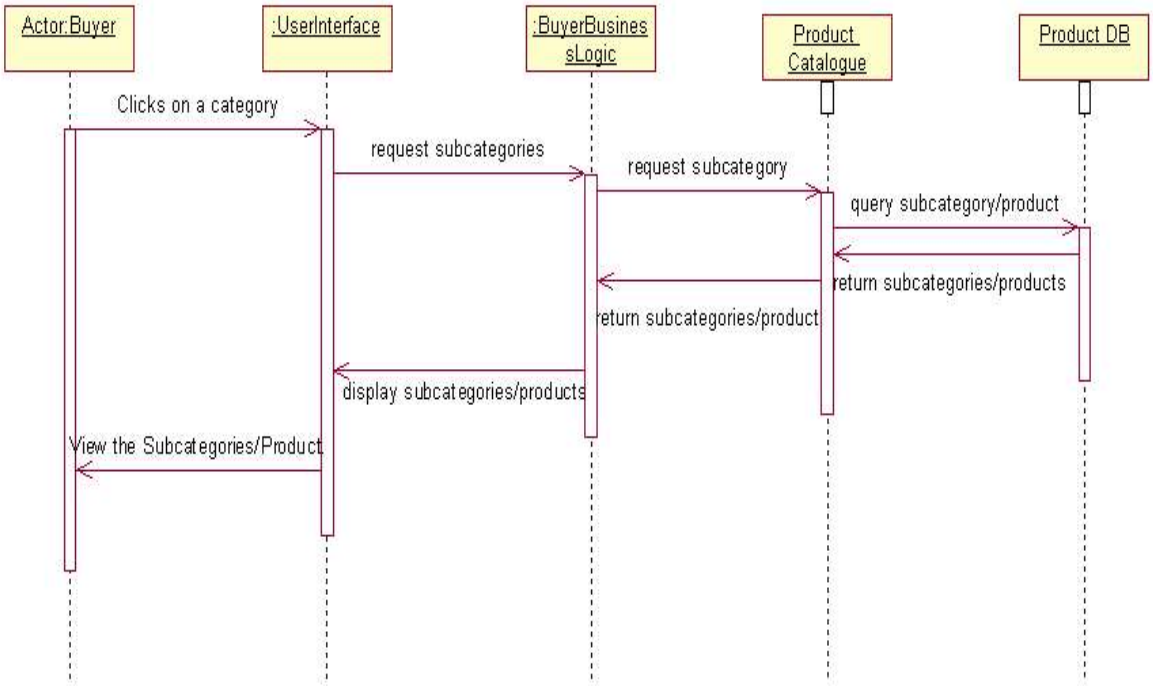


Fig: Sequence Diagram for Product Catalogue

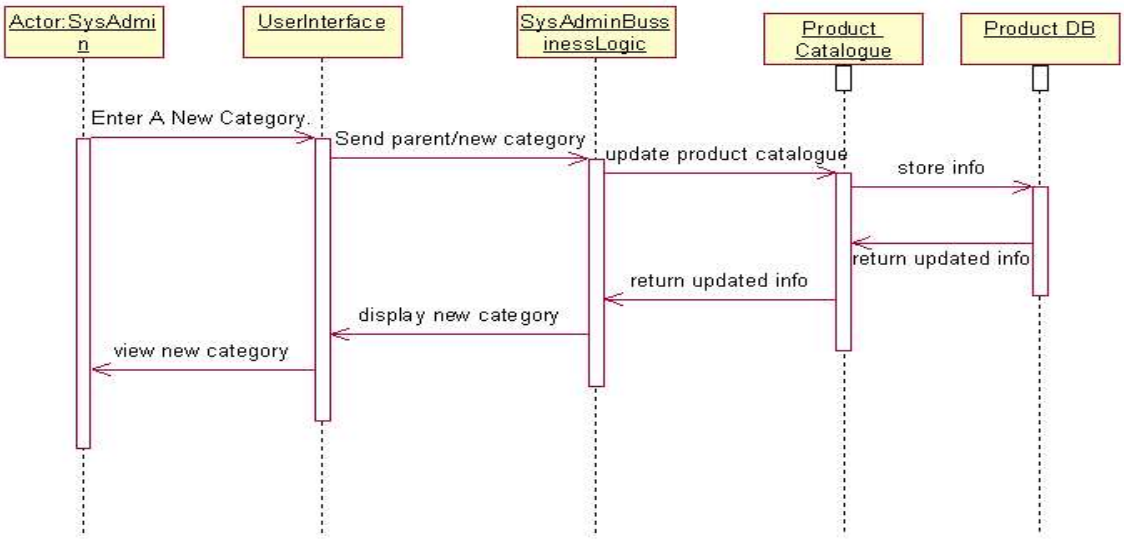


Fig: Sequence diagram for add sub category/product to product catalogue

9. Modeling of State chart Diagram:

A Statechart diagram describes a state machine. 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.

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. The most important purpose of Statechart diagram is to model lifetime of an object from creation to termination.

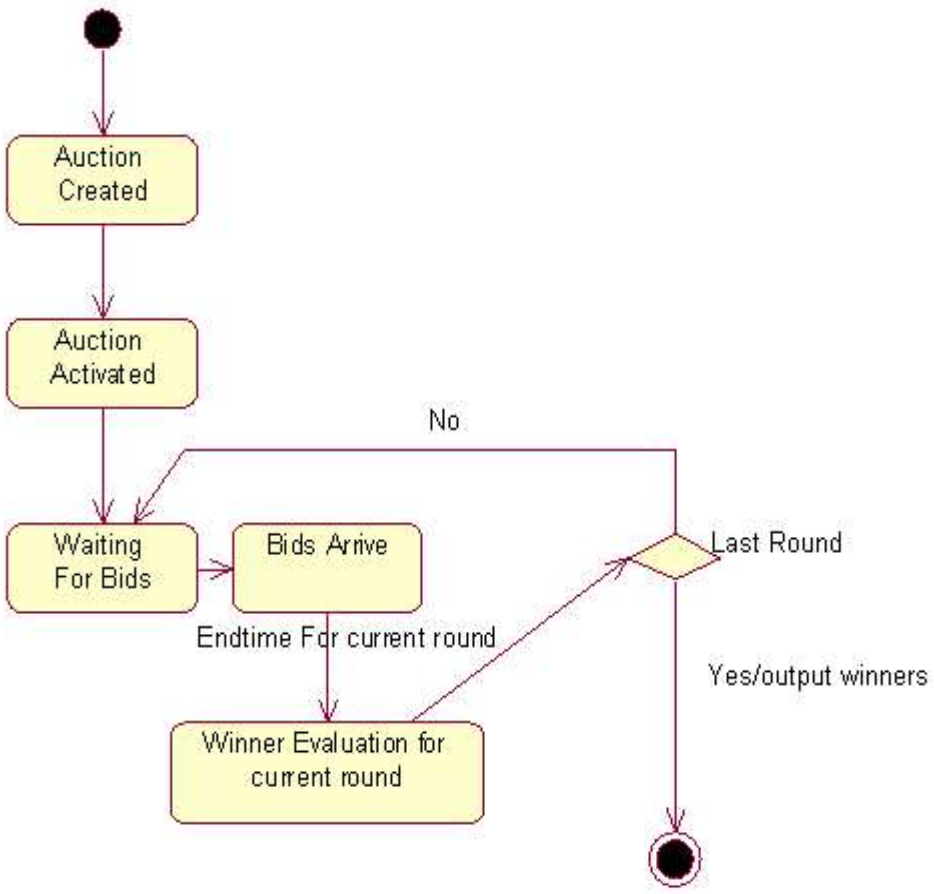
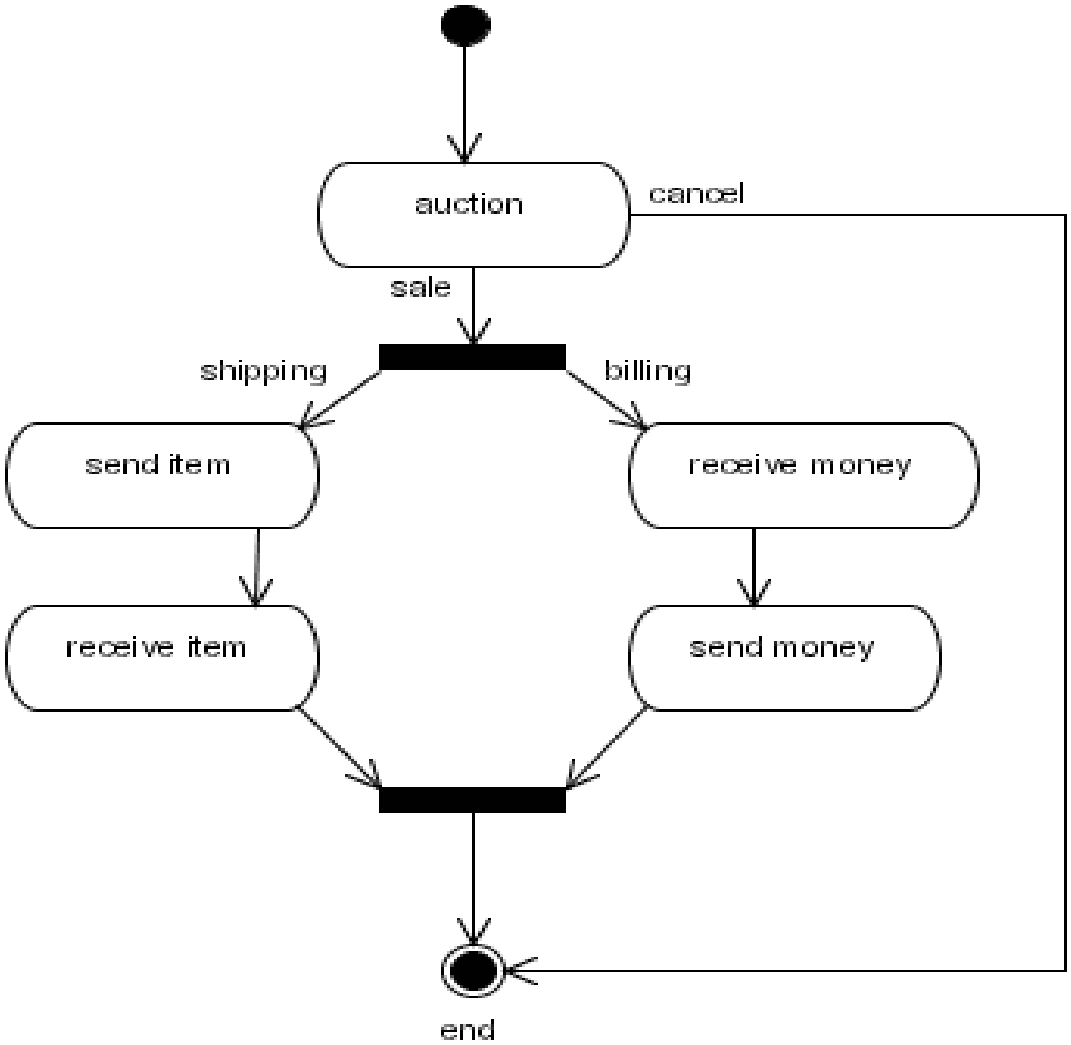


Fig: Statechart Diagram for an Application

10. Modeling of Activity Diagram:

Activity diagram captures the dynamic behavior 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 the 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 the activity diagram is the message part.



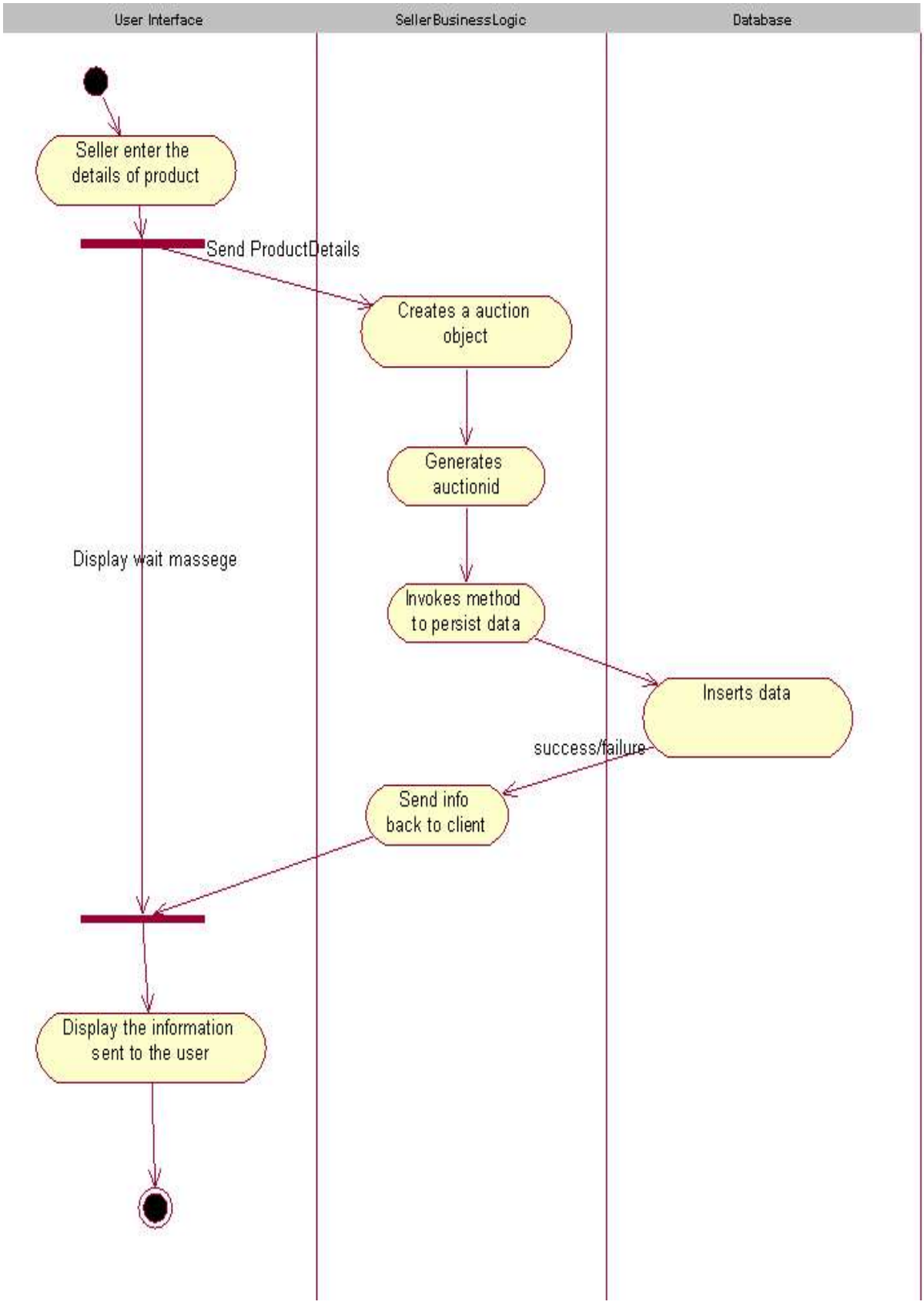


Fig: Activity Diagram for an Auction Application

11. Modeling of Component Diagram:

Component diagrams are used to model the physical aspects of a system. Component diagrams are used to visualize the organization and relationships among components in a system. These diagrams are also used to make executable systems.

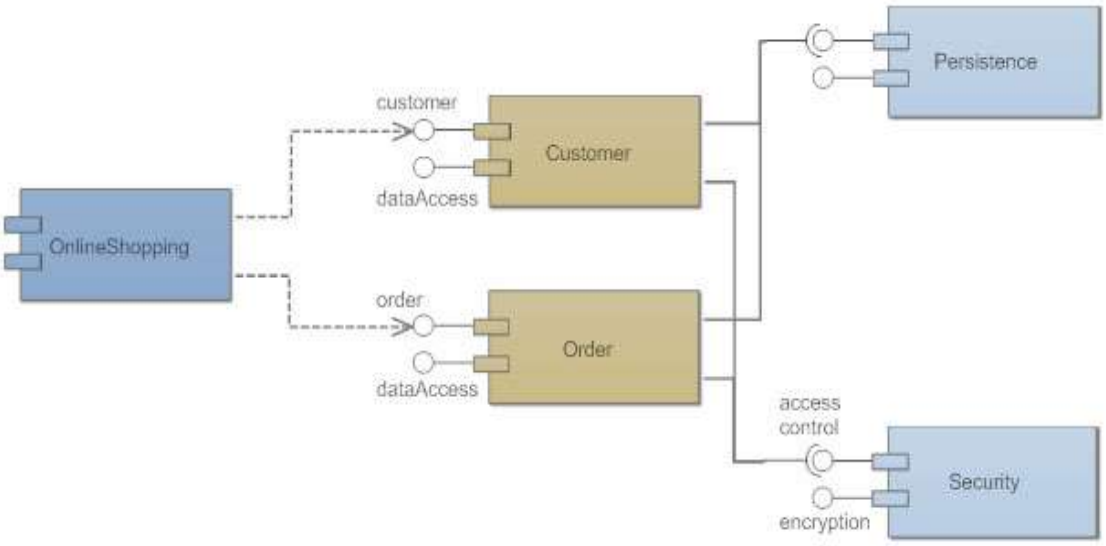


Fig: Component Diagram for an Auction Application

12. Modeling of Deployment Diagram:

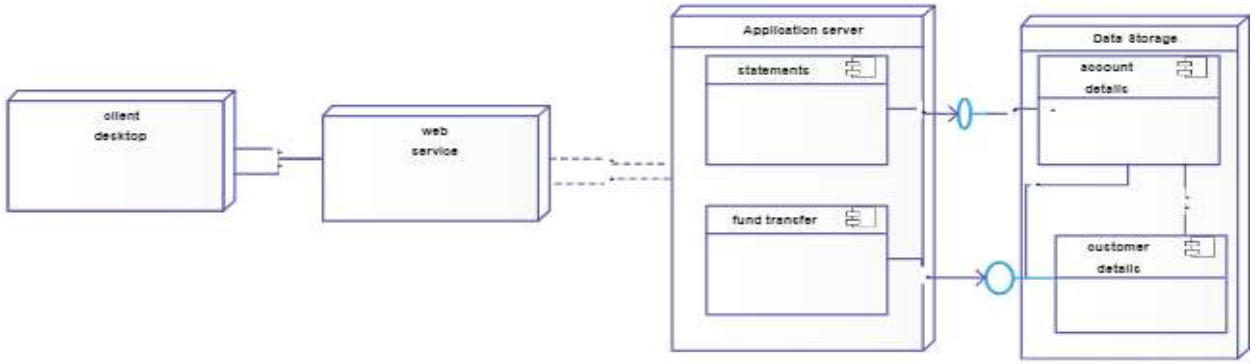
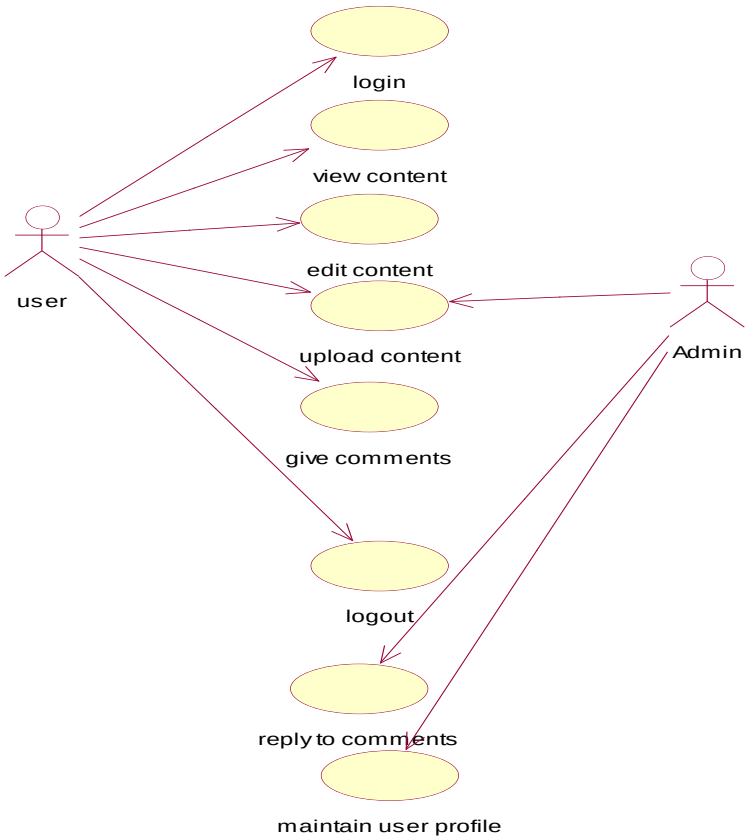


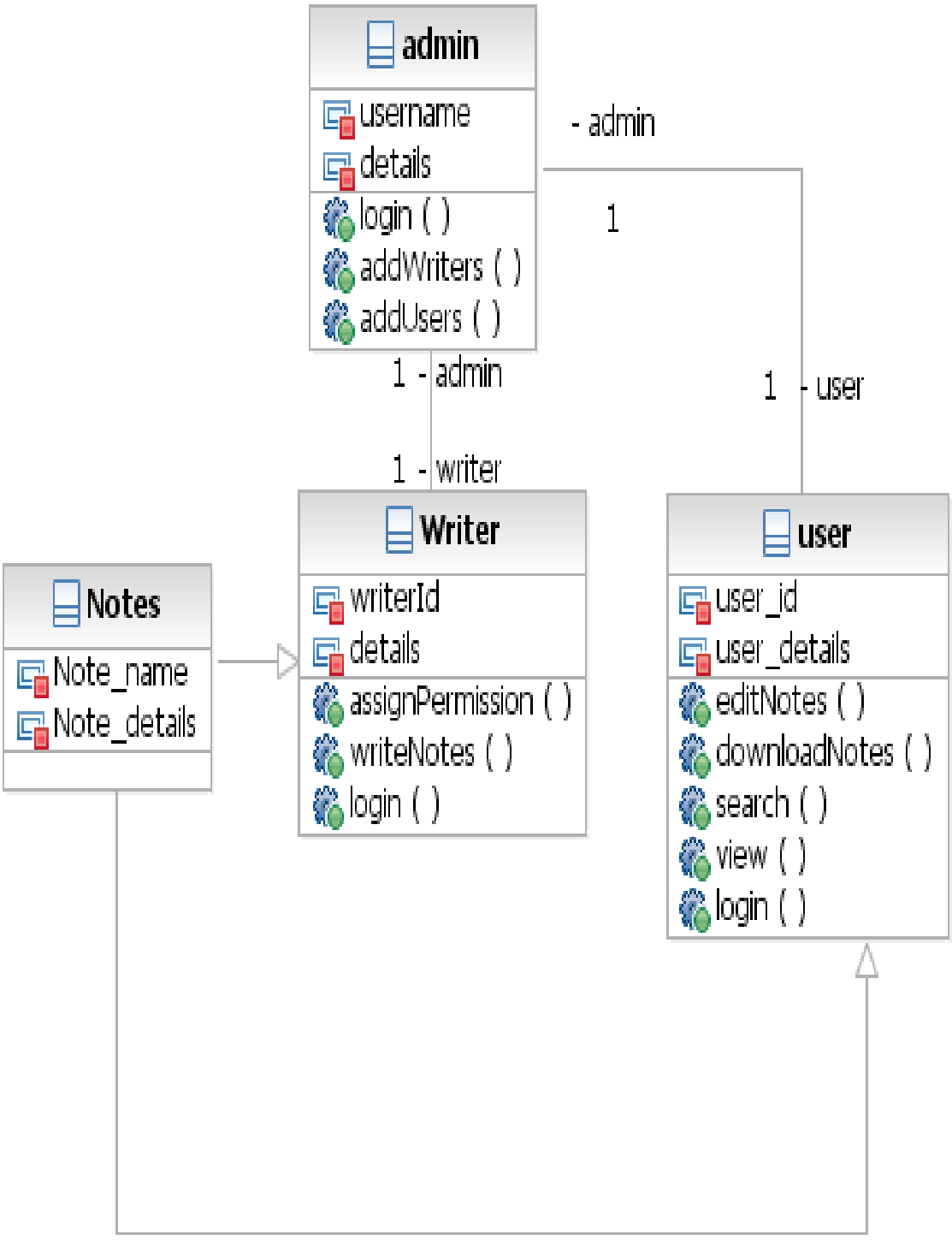
Fig: Deployment Diagram for an Auction Application

Project - VIII : A Notes and File Management System : In the course of one's student years and professional career one produces a 1 lot of personal notes, documents. All these documents are usually kept 1 on papers or individual files on the computer. Either way the bulk of the information is often erased corrupted and eventually lost. The goal of this 1 project is to build a distributed software application that addresses this "problem. The system will provide an interface to create, organize and manage I personal notes through the Internet for multiple users. The system will also allow users to collaborate by assigning permissions for multiple users to view and edit notes.

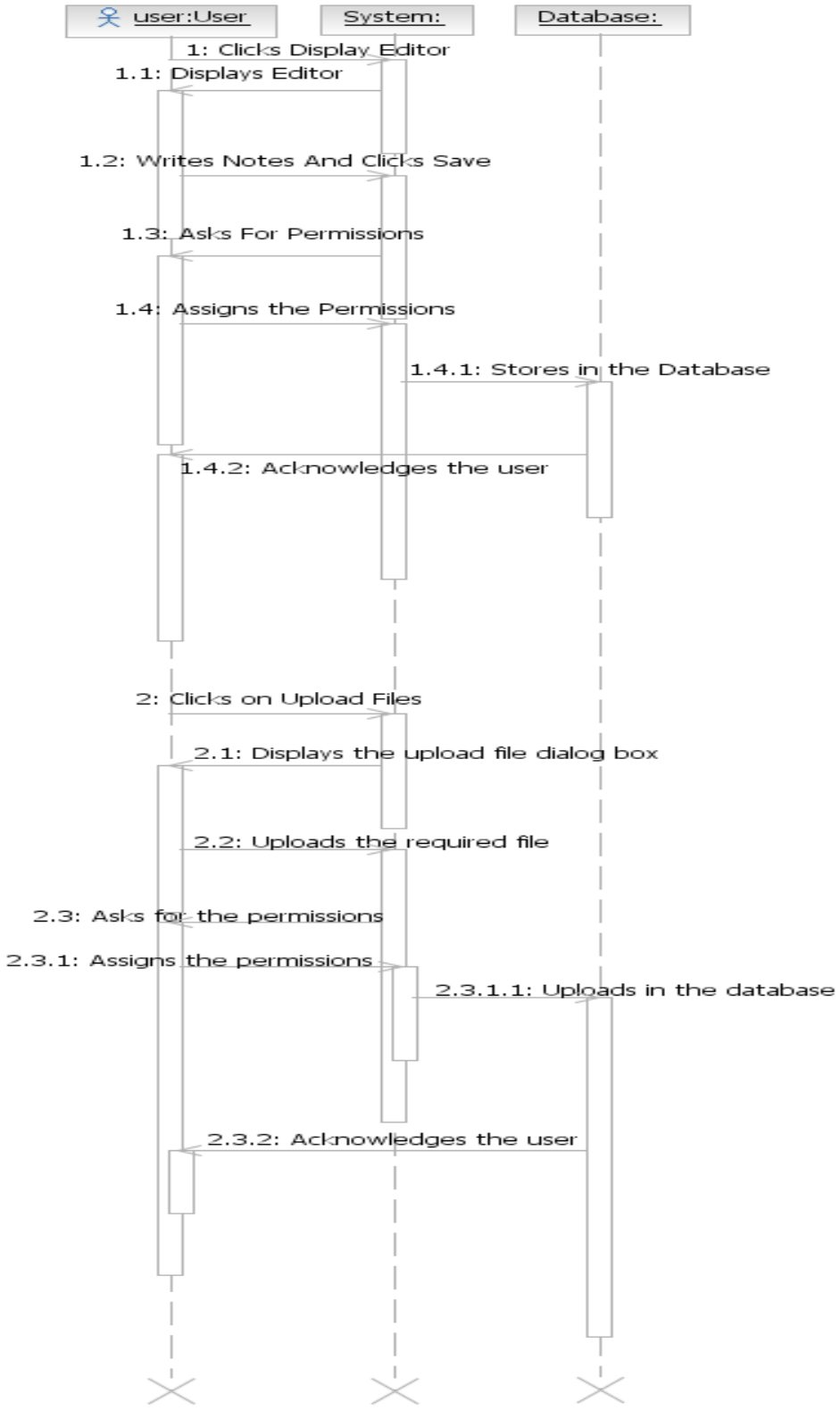
Usecase Diagram:



Class Diagram:



Sequence Diagram:



Project-IX: Library Management System: Library Management System(LMS): The goal is to enable students and librarians to easily access and manage the library and run it smoothly. Each physical library item - book, tape cassette, CD, DVD, etc. could have its own item number. To support it, the items may be barcoded. The purpose of barcoding is to provide a unique and scannable identifier that links the barcoded physical item to the electronic record in the catalog. Barcode must be physically attached to the item, and barcode number is entered into the corresponding field in the electronic item record. Barcodes on library items could be replaced by RFID tags. The RFID tag can contain item's identifier, title, material type, etc. It is read by an RFID reader, without the need to open a book cover or CD/DVD case to scan it with barcode reader.

Detailed Description: A college library management is a project that manages and stores books information electronically according to students needs. The system helps both students and library manager to keep a constant track of all the books available in the library. It allows both the admin and the student to search for the desired book. It becomes necessary for colleges to keep a continuous check on the books issued and returned and even calculate fine. This task if carried out manually will be tedious and includes chances of mistakes. These errors are avoided by allowing the system to keep track of information such as issue date, last date to return the book and even fine information and thus there is no need to keep manual track of this information which thereby avoids chances of mistakes. Thus this system reduces manual work to a great extent allows smooth flow of library activities by removing chances of errors in the details.

The library, contains tens of thousands of books and members which must be organized in order to prevent chaos. As our technology improves, we start to see more and more the enormous amount of help which we can receive from computers. Therefore, the logical and simple solution to this organization problem is to computerize the whole library system.

For this purpose, we built a system which can help all the libraries in the world: A "Computerized Library System" . Our system has a GUI (graphical user interface) on the

internet which allows all of the members , both readers and librarians, a full function access to the library system through the Internet.

Goal: The goal of this project is to create a system for library management. The system will allow performance of the actions needed in order to manage the library in a simple and comfortable way. The actions will include addition/removal of books, addition/removal of members, member and book searches, and much more. The system in parallel to the user actions keeps a basic security level which prevents access or modifications of data by users which don't posses the proper permissions.

The system will support two different types of users. Here are the types and the actions available for each type:

Functional Requirements:

Librarians

- Addition/Removal of a book from the library
- Addition/Removal of a user from the library
- Search for books or members
- Addition/Removal of users from Book- Order list
- Blocking of users who return books late
- Rent a book
- Rent-out a book to a user

Readers

- Searching of books or users.
- Addition of self to Book- Order list
- Rent a book

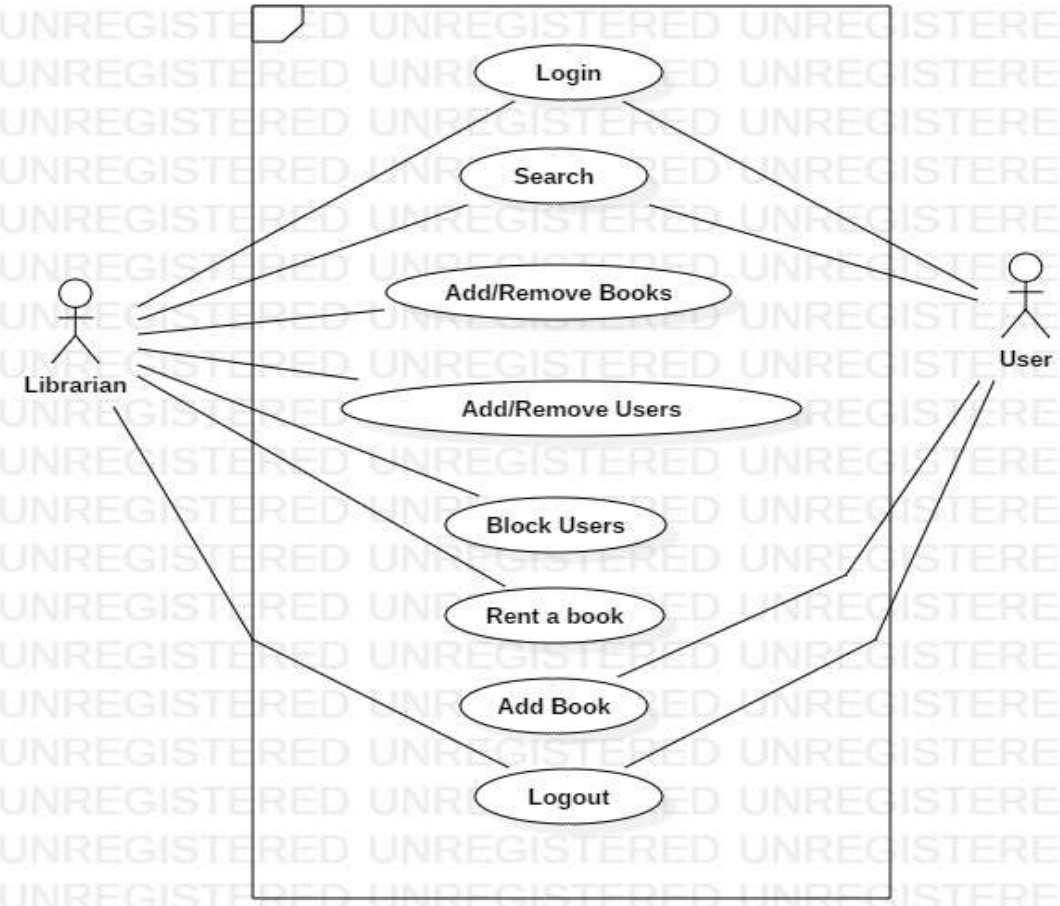
Comment: For each user (reader/librarian) exists a unique Id and password, this allows an unambiguous recognition of the user, so that the system can prevent from users access to information which they are not allowed to access.

Comment: For each user (reader/librarian) exists a unique Id and password, this allows an unambiguous recognition of the user, so that the system can prevent from users access to information which they are not allowed to access.

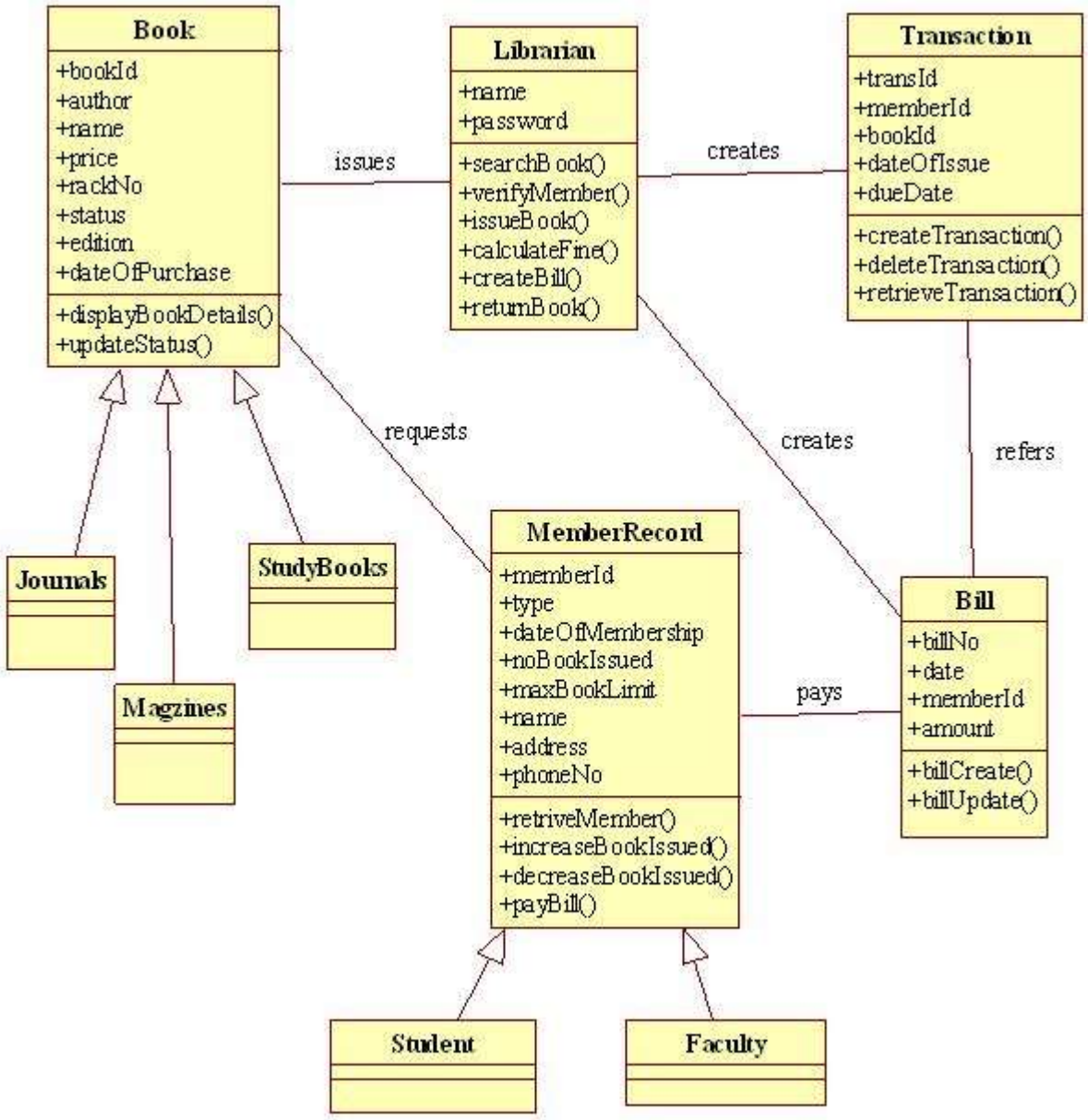
Non Functional Requirements:

- Reliability
- Security
- Integrity
- Portability
- Testability

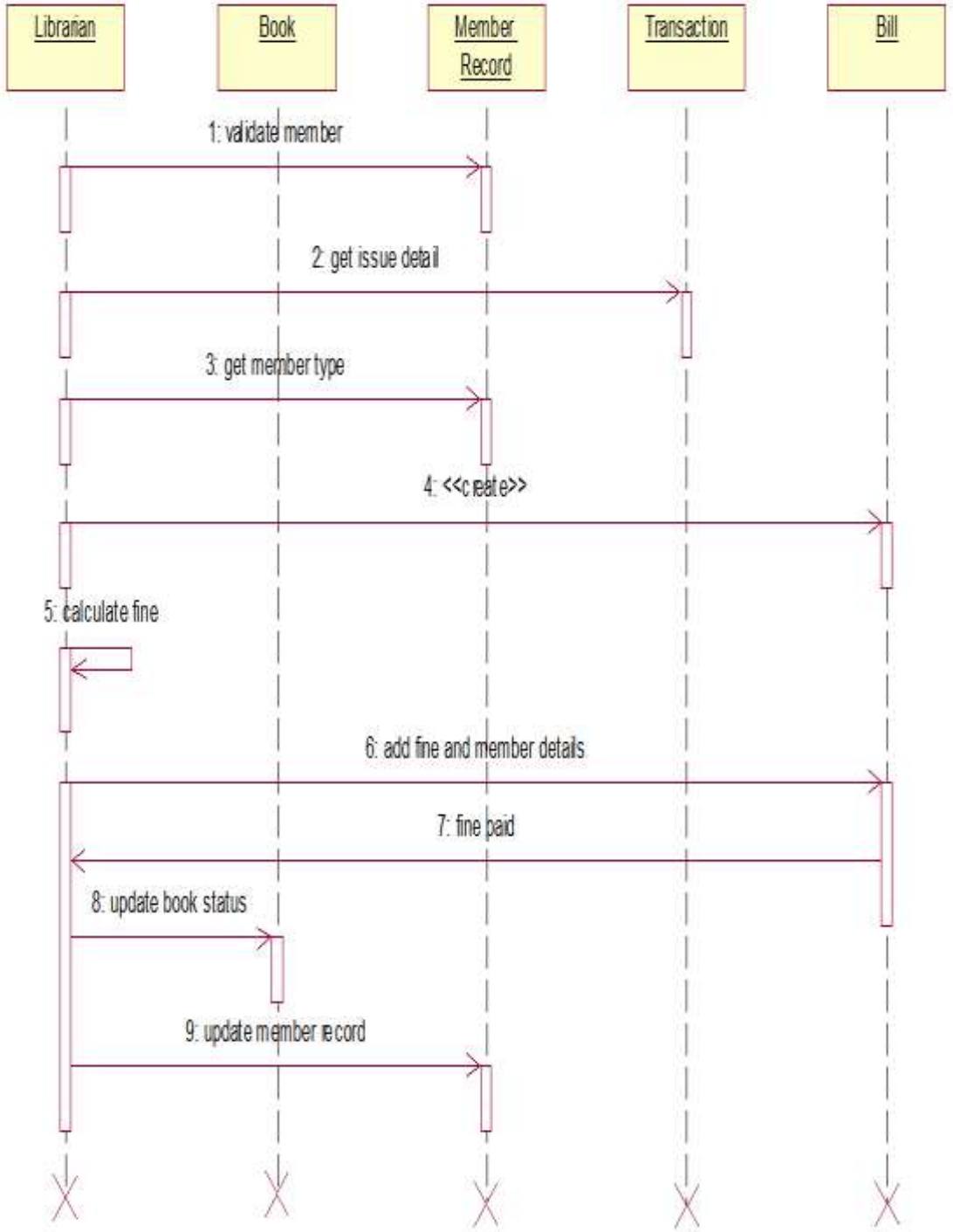
1. Usecase Diagram:



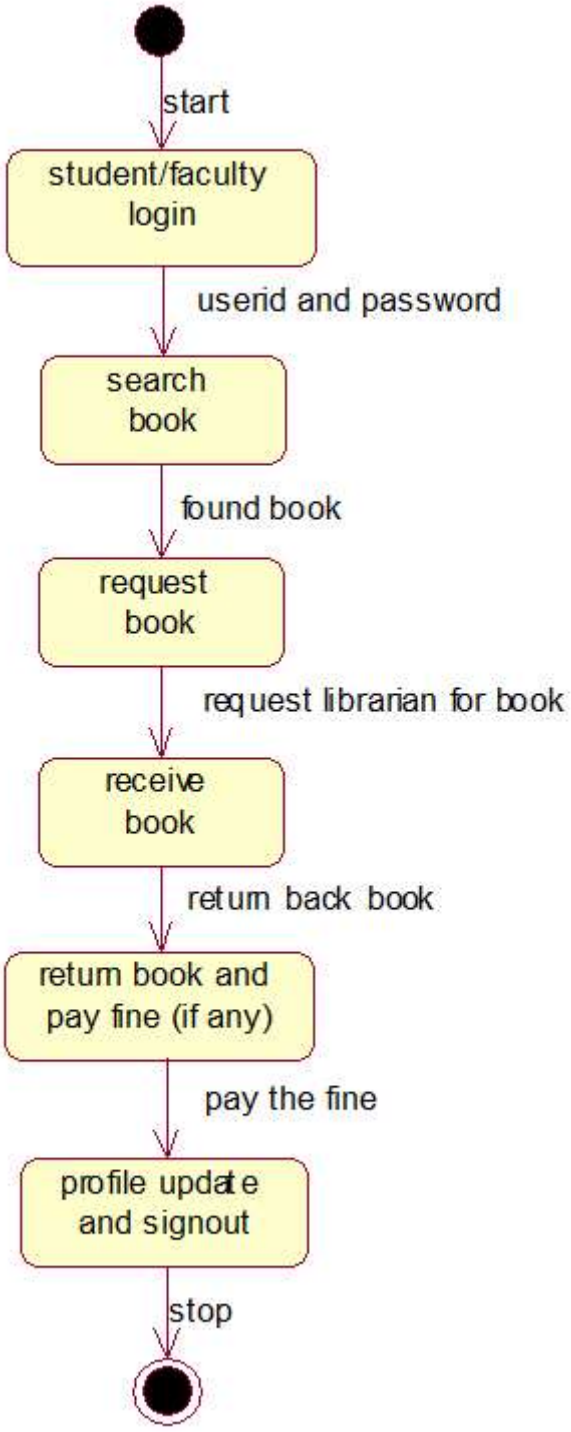
2. Class Diagram:



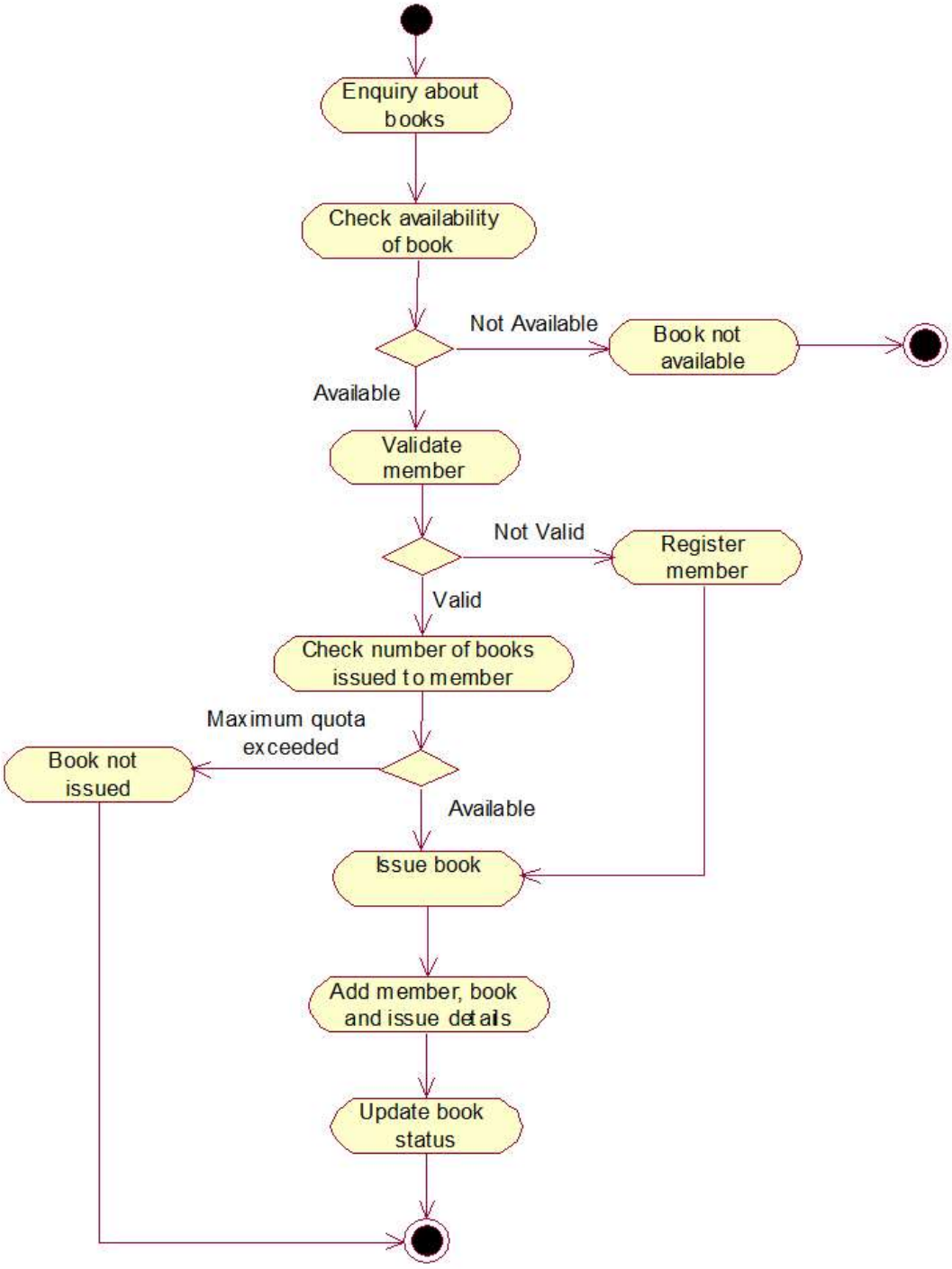
3. Sequence Diagram:



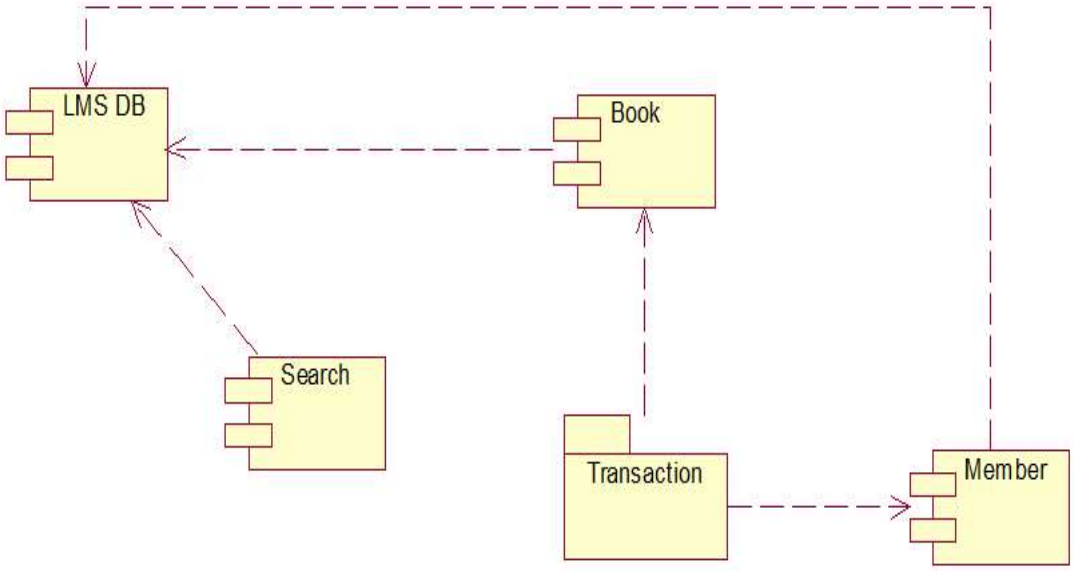
4. State chart Diagram:



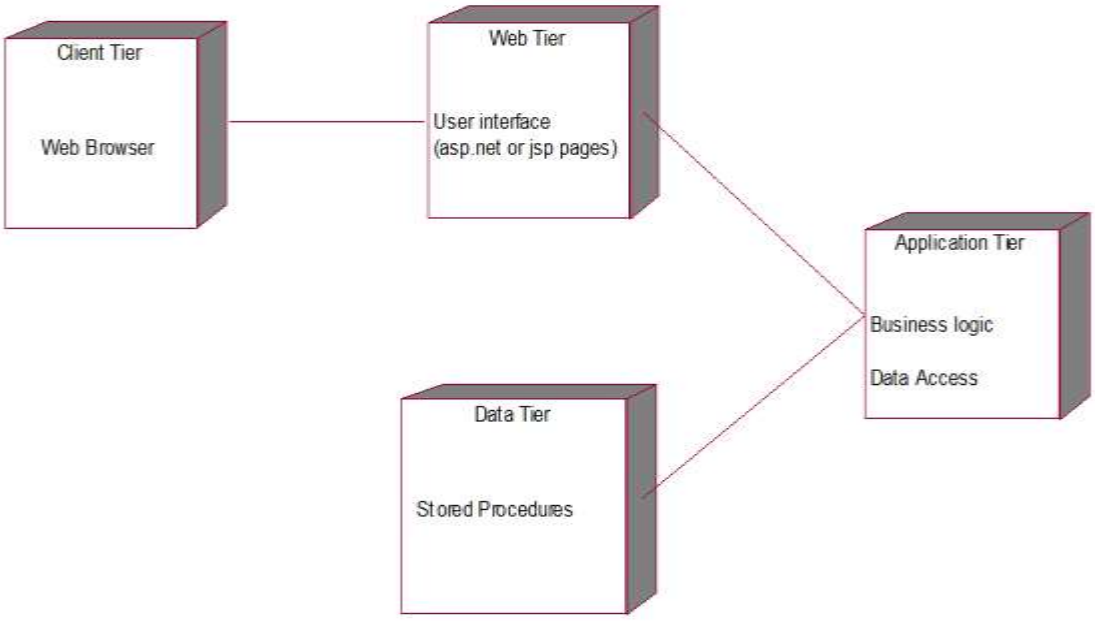
5. Activity Diagram:



6. Component Diagram:



7. Deployment Diagram:



Project-X: Hospital Management System: Simulate to show and explain hospital structure, staff, and relationships with patients, and patient treatment terminology.

The purpose of the project entitled as "HOSPITAL MANAGEMENT SYSTEM" is to computerize the Front Office Management of Hospital to develop software which is user friendly simple, fast, and cost effective. It deals with the collection of patient's information, diagnosis details, etc. Traditionally, it was done manually. The main function of the system is register and store patient details and doctor details and retrieve these details as and when required, and also to manipulate these details meaningfully System input contains patient details, diagnosis details, while system output is to get these details on to the screen. The Hospital Management System can be entered using a username and password. It is accessible either by an administrator or receptionist. Only they can add data into the database. The data can be retrieved easily. The data are well protected for personal use and makes the data processing very fast.

1. Detailed Description: The project Hospital Management system includes registration of patients, storing their details into the system, and also computerized billing in the pharmacy, and labs. The software has the facility to give a unique id for every patient and stores the details of every patient and the staff automatically. It includes a search facility to know the current status of each room. User can search availability of a doctor and the details of a patient using the id. The Hospital Management System can be entered using a username and password. It is accessible either by an administrator or receptionist. Only they can add data into the database. The data can be retrieved easily. The interface is very user-friendly. The data are well protected for personal use and makes the data processing very fast. Hospital Management System is powerful, flexible, and easy to use and is designed and developed to deliver real conceivable benefits to hospitals.

Hospital Management System is designed for multispeciality hospitals, to cover a wide range of hospital administration and management processes. It is an integrated end-to-end Hospital Management System that provides relevant information across the hospital to support effective decision making for patient care, hospital administration and critical financial accounting, in a seamless flow. Hospital Management System is a software product suite designed to improve the quality and management of hospital management in the areas of

clinical process analysis and activity-based costing. Hospital Management System enables you to develop your organization and improve its effectiveness and quality of work. Managing the key processes efficiently is critical to the success of the hospital helps you manage your processes.

Functional Requirements:

Receptionist:

- Give Admition
- Give Doctors appointment
- Test appointment
- Bed allotment

Staff/Clerk:

- Login
- Draw salary
- Give Patient payment information

Doctor:

- Prescribe test
- Login
- Draw Salary

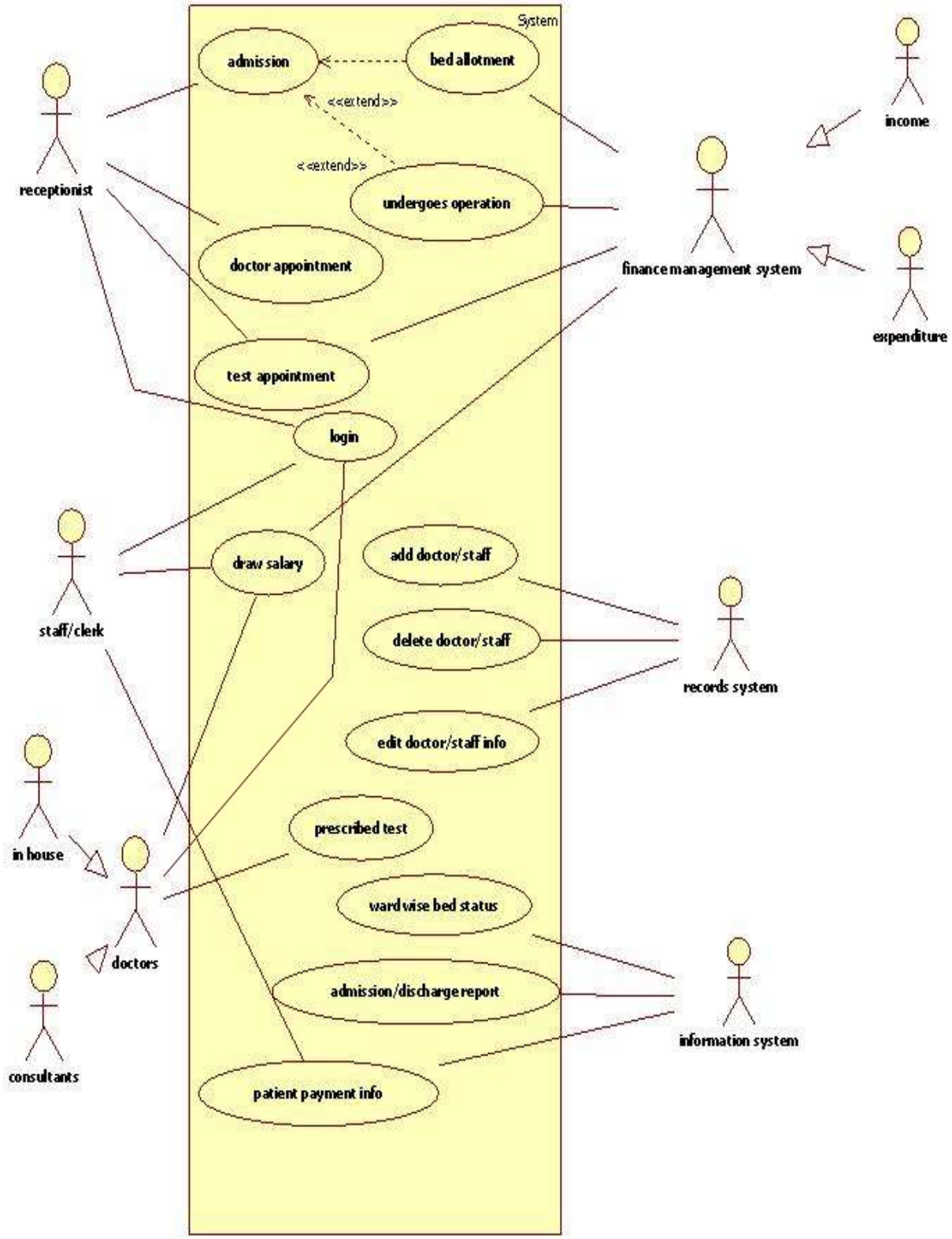
Record System:

- Add Doctor/Staff
- Delete Doctor/Staff
- Edit Doctor/Staff

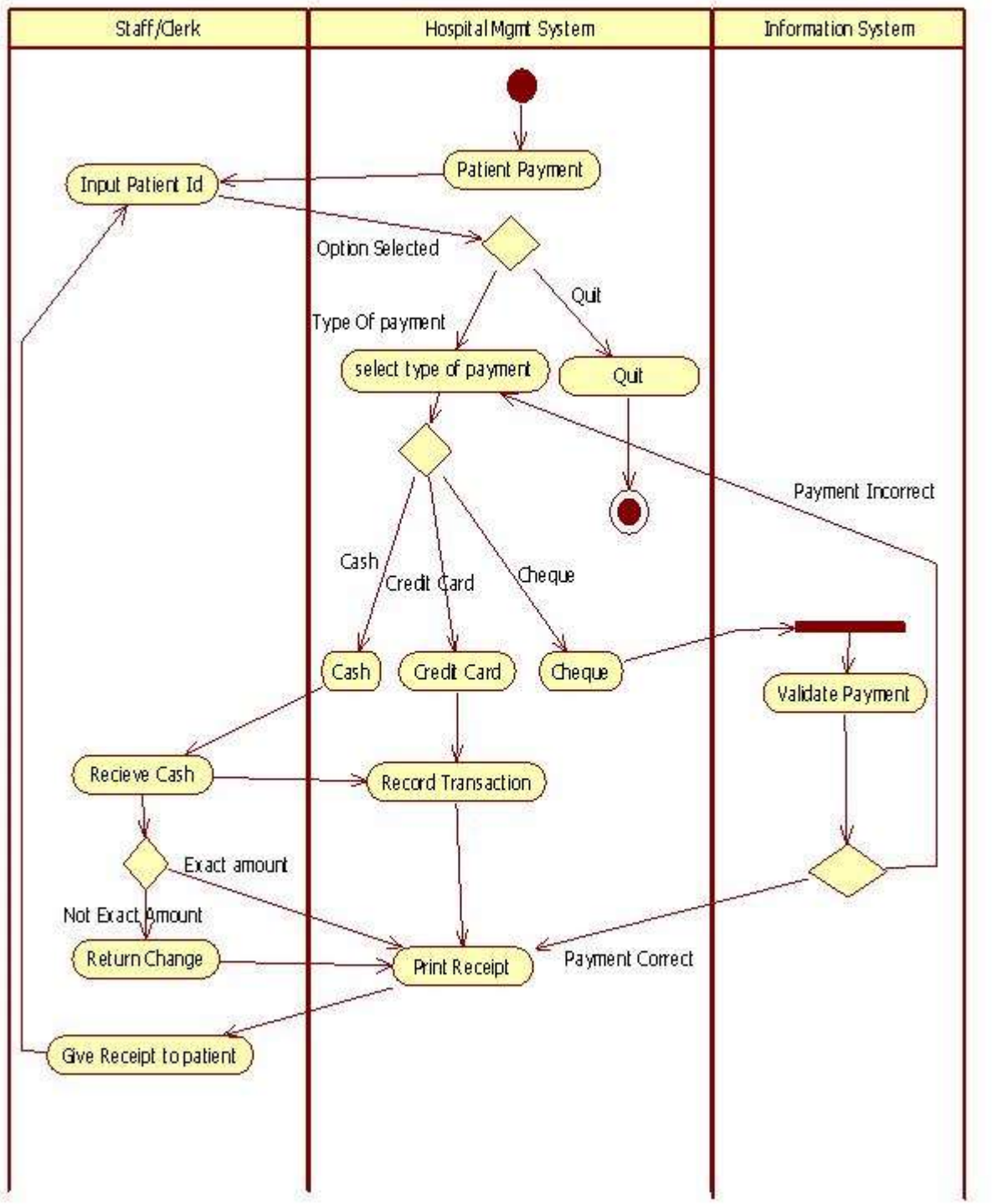
Finance Management System:

- Bed allotment
- Undergo operation
- Test appointment
- Draw Salary

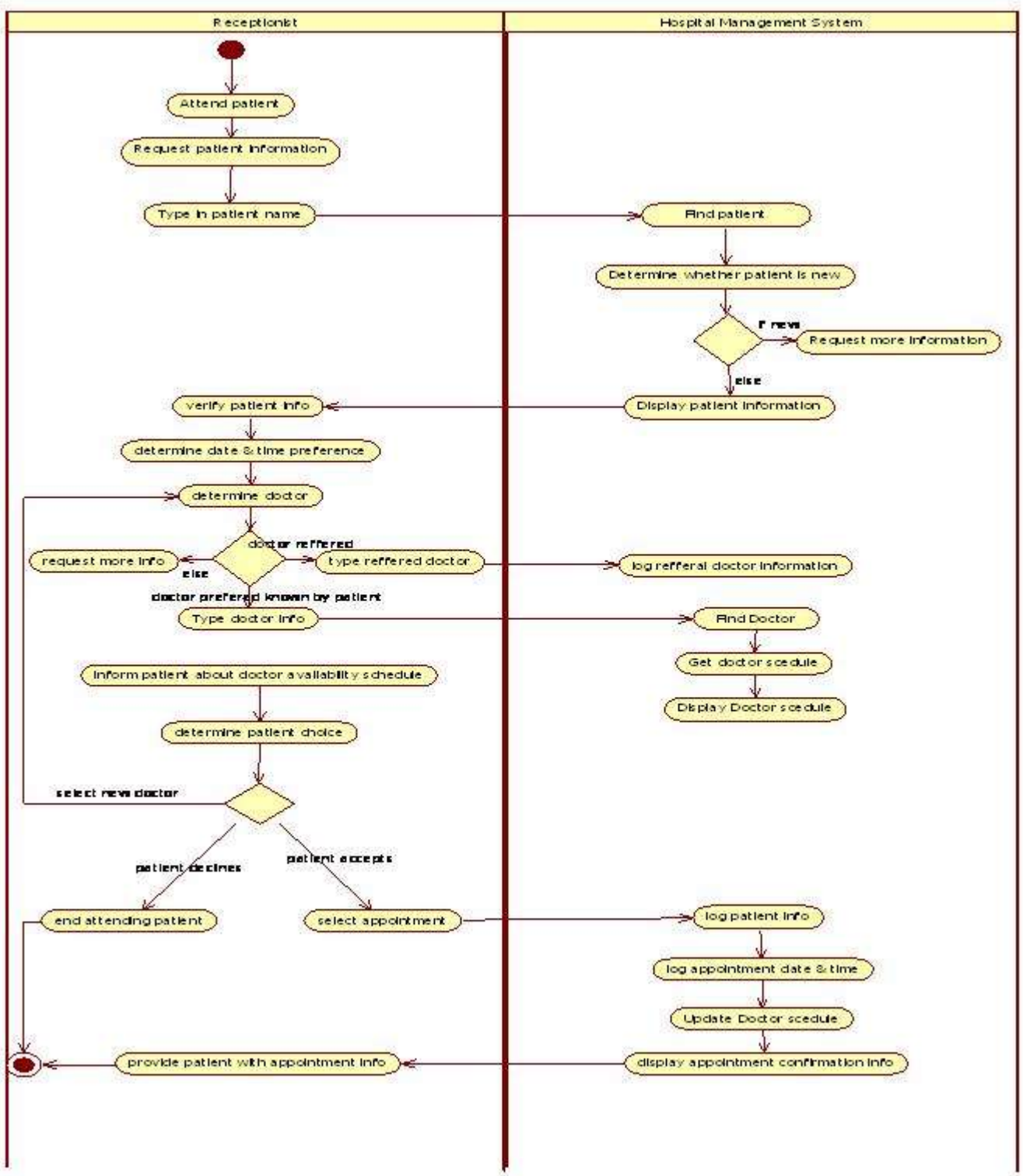
1. Use Case Diagram:



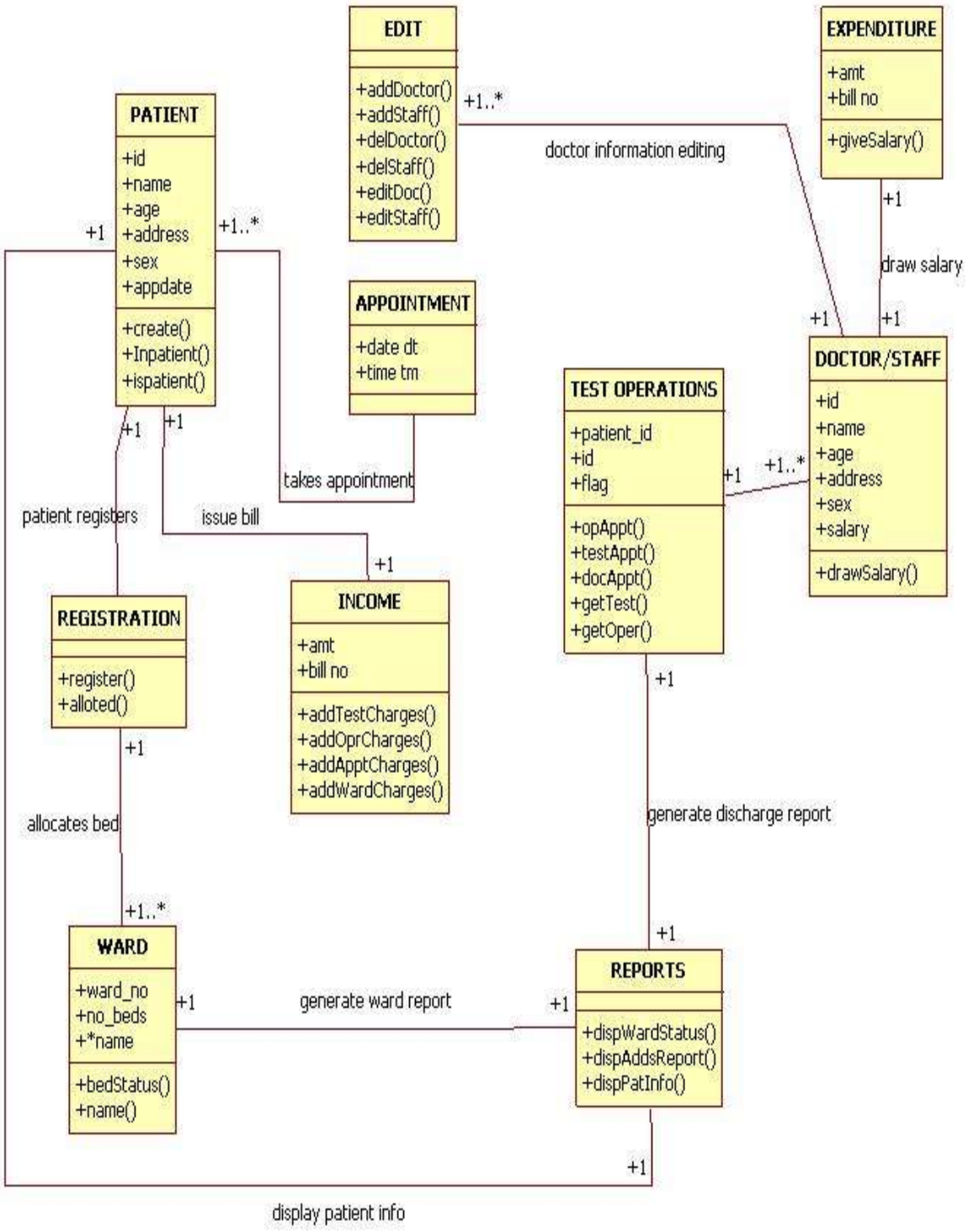
2. Activity Diagram:



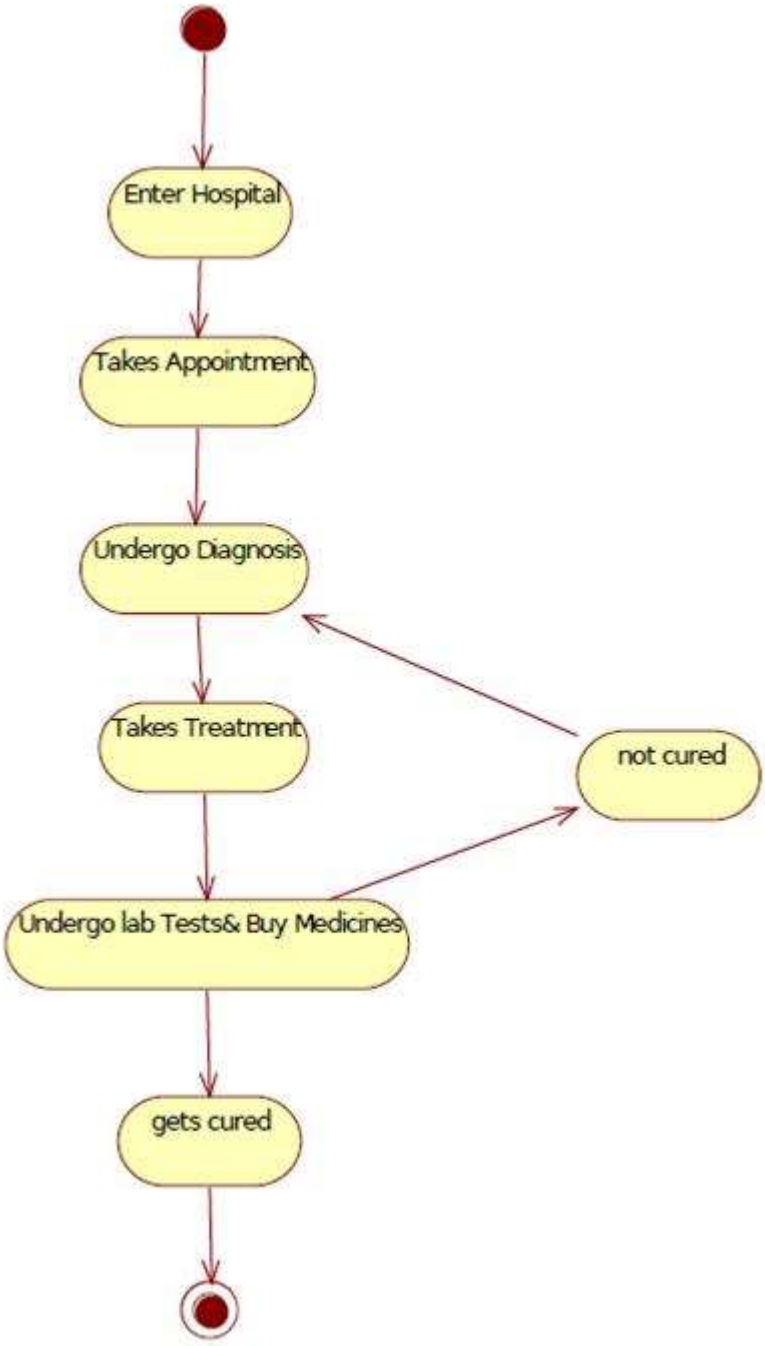
2.1. Activity Diagram2:

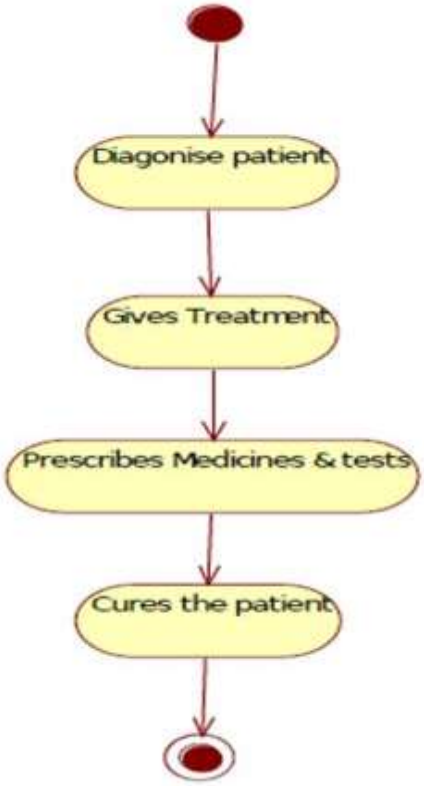


3. Class Diagram:



4. State Chart Diagram:

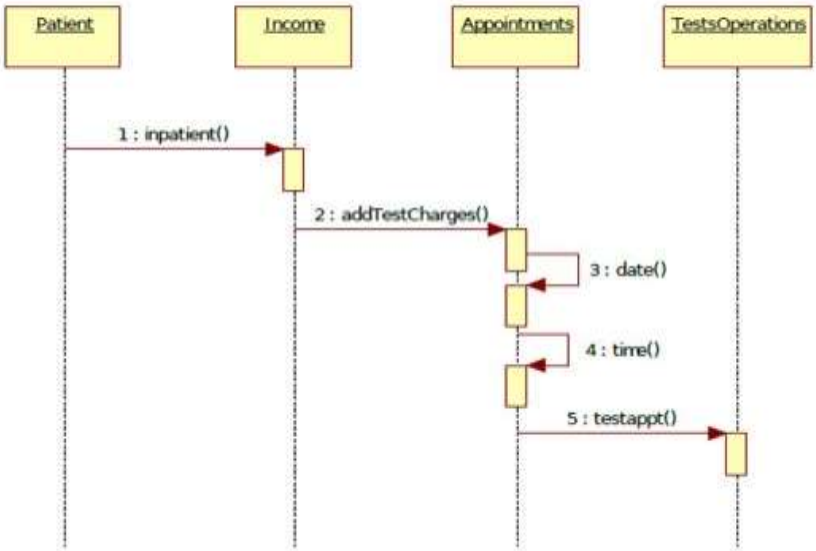



Diagonise patient
Gives Treatment
Prescribes Medicines & tests
Cures the patient

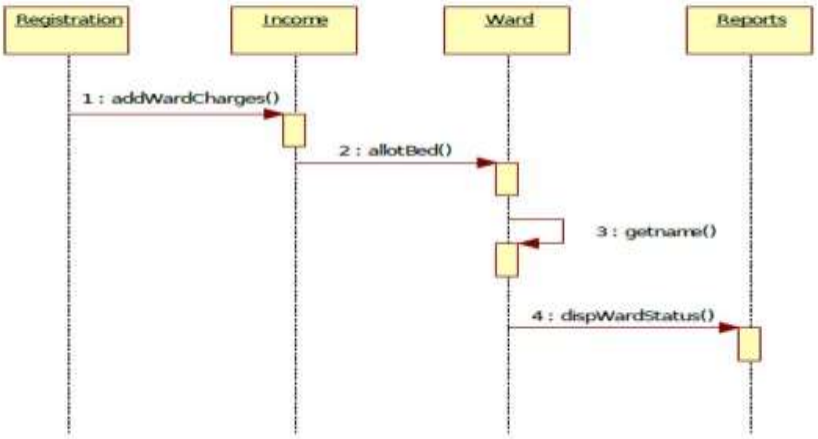

Takes Details os patient
Checks availabilty of doctor
gives appointment
gives bill
takes bill amount

5. Sequence Diagram:

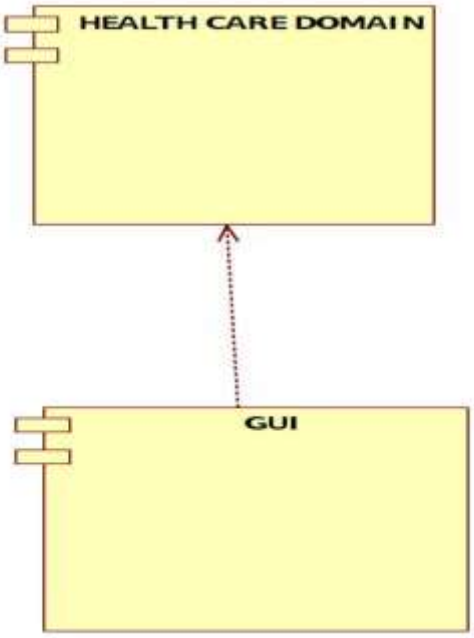
TESTS APPOINTMENTS:



BED ALLOTMENT:



6. Component Diagram:



7. Deployment Diagram: