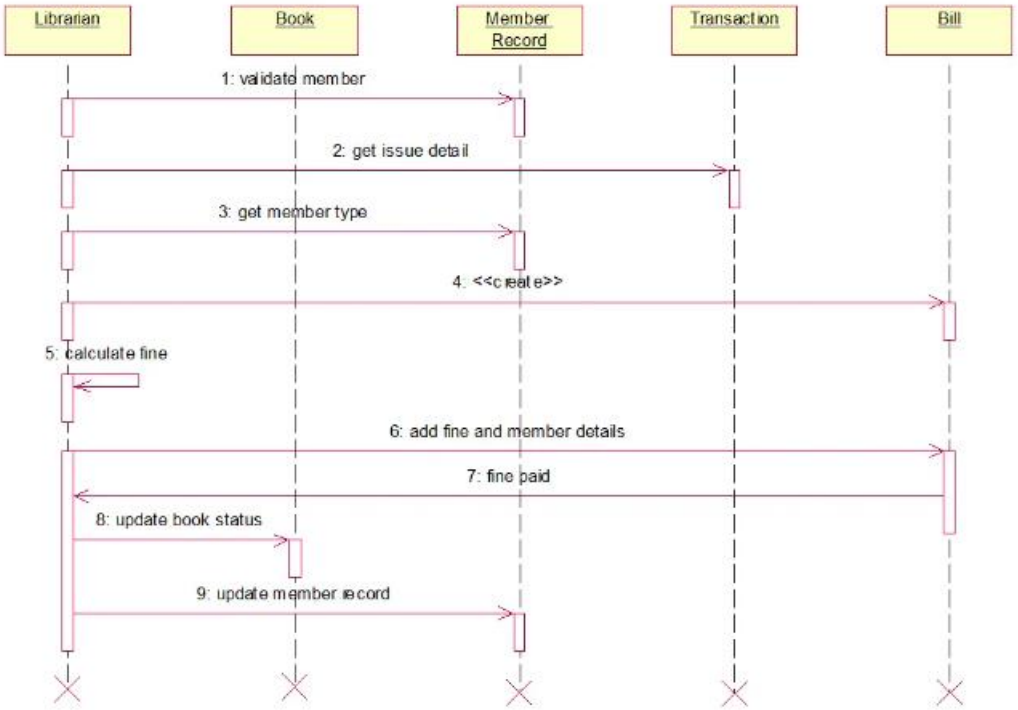
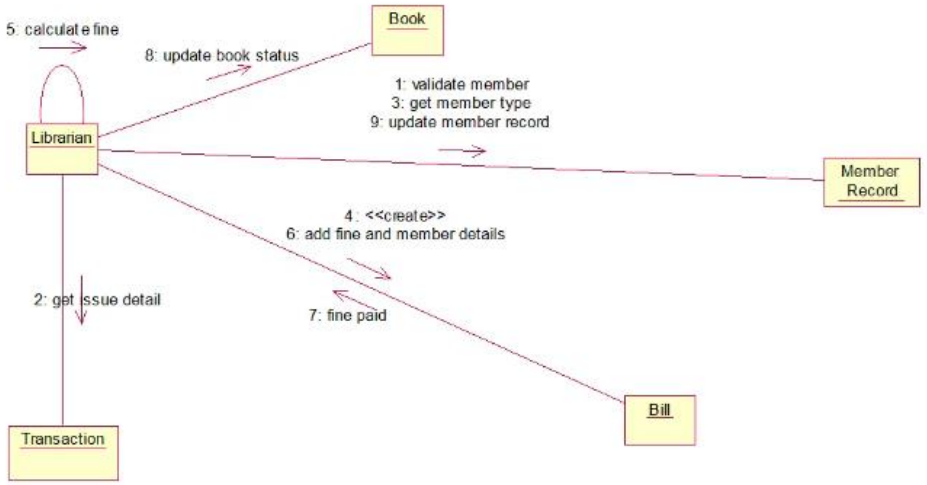
1. Interaction diagrams for unified library application.

Sequence diagram



Collaboration diagram



2. Collaboration and sequence diagram and its differences.

Key	Sequence Diagram	Collaboration diagram
Definition	A Sequence diagram is a type of UML diagram in which the main representation is of the sequence of messages flowing from one object to another; also main emphasis is on representing that how the messages/events are exchanged between objects and in what time-order.	A Collaboration diagram is a type of UML diagram in which the main representation is of how one object is connected to another implementing the logic behind these objects with the use of conditional structures, loops, concurrency, etc.
Main focus	The focus is on representing the interaction between different objects by pictorial representation of the message flow from one object to another object. It is time ordered that means exact interactions between objects is represented step by step.	The primary objective is to represent the structural organization of the system and the messages that are sent and received.
Type	A Sequence diagram models the sequential logic, ordering of messages with respect to time, so it is categorized as Dynamic modelling diagram.	A Collaboration diagram mainly represents the organization of a system, so it is not classified as Dynamic modelling diagram.
Use case	Sequence diagram is used to describe the behavior of several objects in a particular single use case with implementation of all possible logical conditions and flows.	Collaboration diagrams are used to describe the general organization of system for several objects in several use cases.

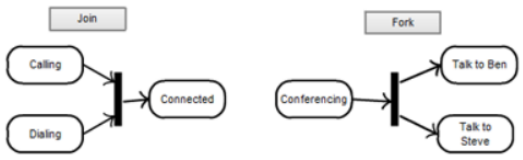
3. Describe fork and join in activity diagram with example.

Fork / Join

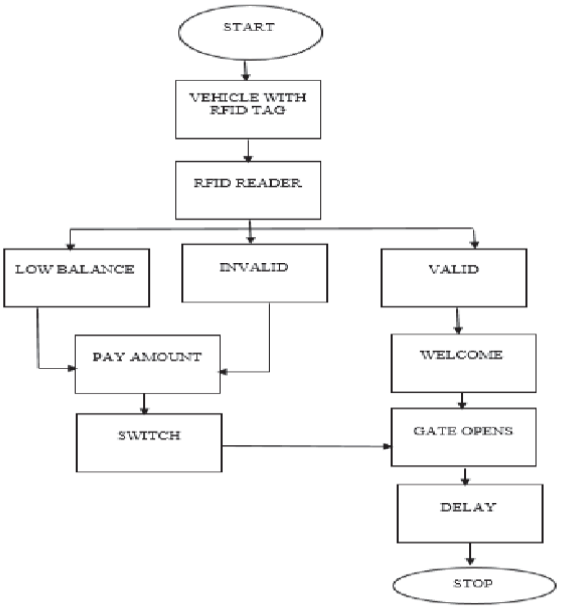
A Fork notation in a **UML Activity Diagram** is a control node that splits a flow into multiple concurrent flows. This will have one incoming edge and multiple outgoing edges. A join node is a control node that synchronizes multiple flows. This will have multiple incoming edges and one outgoing edge.

Fork vertices in the **UML Statechart Diagram** serve to split an incoming transition into two or more transitions terminating on orthogonal target vertices. The segments outgoing from a fork vertex must not have guards or triggers. Join vertices serve to merge several transitions emanating from source vertices in different orthogonal regions. The transitions entering a join vertex cannot have guards or triggers.

You can easily create them online using our activity diagram tool.



4. Activity diagram to show business process of Toll Plaza.

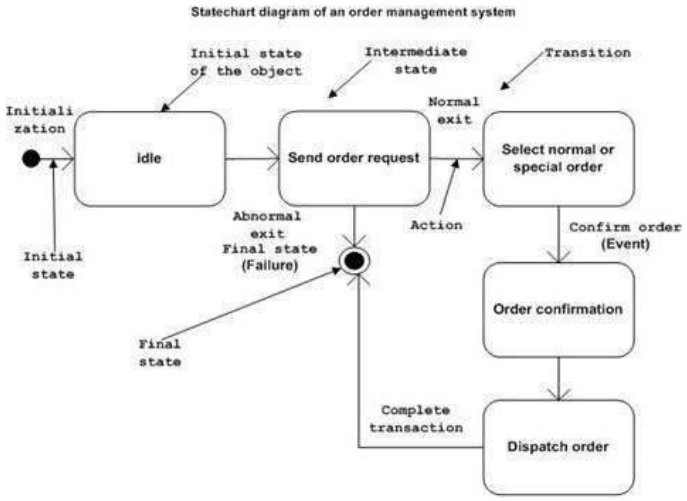


5. State chart diagrams with example.

A state chart diagram is a type of UML diagram that is used to model the dynamic behavior of a system. It shows how the state of an object changes over time in response to events.

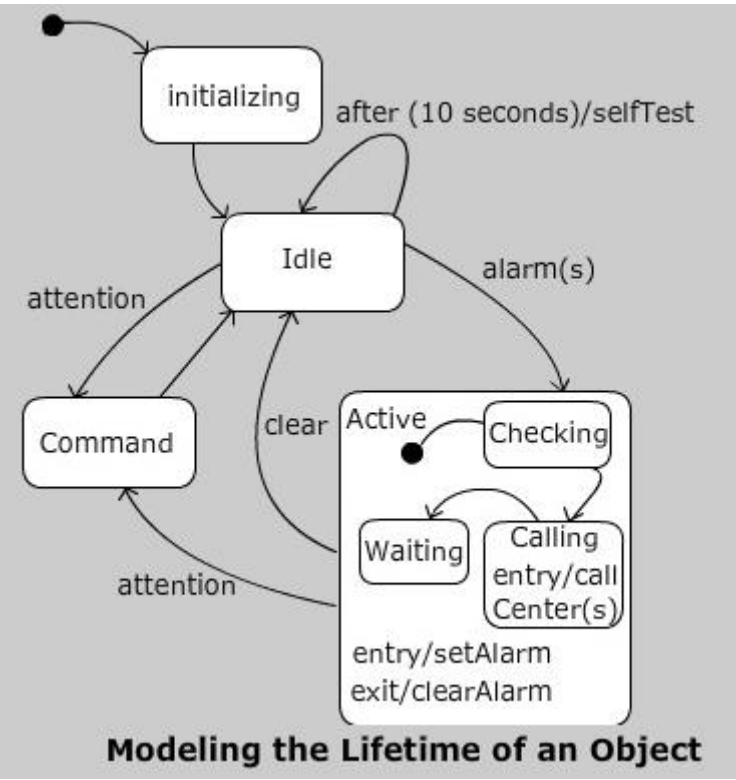
A state chart diagram is made up of states, events, and transitions.

- States represent the different conditions that an object can be in. For example, a light switch can be in the on state, the off state, or the in-between state.
- Events are things that happen to an object that can cause it to change state. For example, a light switch might be turned on by a person or by a timer.
- Transitions show how an object moves from one state to another. For example, a light switch might move from the off state to the on state when it is turned on by a person.



6. Common modelling mechanisms of state chart diagram.

- A class, a use case, or the system as a whole can be modeled as a state machine
- If the context is a class or a use case, find out the neighboring classes (which may include any parents of a class) and any classes that are accessible by associations or dependencies
- These neighbors are candidate targets for actions and hence, may figure in guard conditions
- Limit the focus to one behavior of the system, if the context is the system as a whole
- Including all objects is unpractical and counterproductive even if they participate



- Set up the initial and the final states of the object
- Provide the pre and the post conditions of the initial and the final states that determine the rest of the model
- Choose the events to which the object may respond
- If the events are already specified, they can be found in the object's interfaces. Otherwise, consider the objects may be dispatched by that a
- Identify the states the object may be in, from the initial to final state
- Associate these states using the transitions triggered by the appropriate events
- Add actions to these transitions
- Identify any entry or exit actions
- Expand these states by using substates wherever necessary

- Verify that all the events specified in the state machine match the events expected by the interface of the object and vice versa. Some events may be ignored
- Make sure that all actions mentioned in the state machine are supported by the relationships, methods, and operations of the enclosing object
- Trace through the state machine, either manually or by using the tools to check the state machine against expected sequences of events and their responses
- Carefully identify unreachable states and those states in which the machine may not respond
- Make sure that the object's semantics are preserved

7. What is an event and type of events.

In object-oriented design, an event is a significant occurrence that has a location in time and space. Events can be caused by user interaction, such as a button being clicked, or triggered by program logic, such as a timer.

There are four main types of events in object-oriented design:

- User events: These are events that are caused by user interaction, such as a button being clicked or a key being pressed.
- Timer events: These are events that are triggered by a timer, such as a notification that is sent after a certain amount of time has passed.
- Network events: These are events that are caused by network activity, such as a message being received from another computer.
- System events: These are events that are caused by the operating system, such as a window being closed or a file being saved.

Events are important in object-oriented design because they allow objects to communicate with each other. When an event occurs, the object that raised the event sends a notification to all of the objects that are interested in receiving that notification. The objects that receive the notification can then take action in response to the event.

For example, when a button is clicked, the button object raises a click event. The click event is then sent to all of the objects that are interested in receiving click events. The objects that receive the click event can then take action, such as changing the state of the object or displaying a message.

Events are a powerful tool that can be used to create more complex and responsive object-oriented systems. By using events, you can decouple objects from each other and make it easier to change the behavior of your system.

Here are some of the benefits of using events in object-oriented design:

- Events allow for decoupling: Events allow objects to communicate with each other without being tightly coupled. This makes it easier to change the behavior of your system without having to modify the code of every object that is involved.
- Events make it easier to create responsive systems: Events can be used to create event-driven systems. Event-driven systems are responsive to user input and can react to changes in the environment.
- Events make it easier to create reusable code: Events can be used to create reusable code. This is because events can be defined in one place and then used by multiple objects.

If you are interested in learning more about events in object-oriented design, there are many resources available online. You can also find books and articles that discuss the use of events in object-oriented programming.

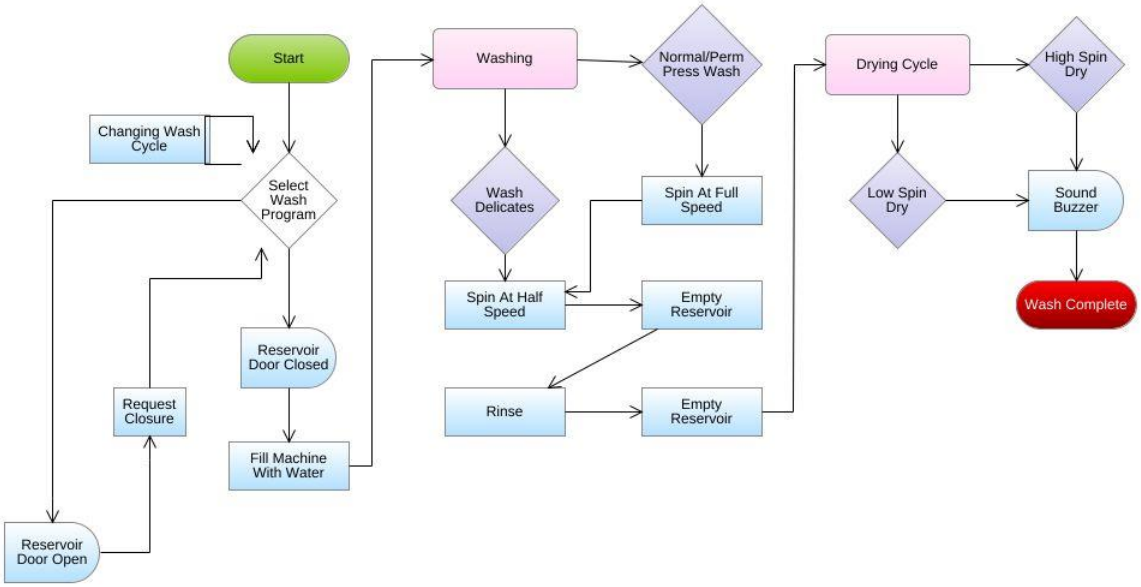
8. What is State Diagram and Draw State diagram for Washing machine.

A state diagram is a type of diagram that shows the different states that an object can be in and the transitions between those states. It is a visual representation of the behavior of an object.

A washing machine can be in a number of different states, such as:

- Off: The washing machine is turned off and is not in use.
- On: The washing machine is turned on and is ready to be used.
- Washing: The washing machine is washing clothes.
- Rinsing: The washing machine is rinsing clothes.
- Spinning: The washing machine is spinning clothes to remove water.
- Done: The washing machine is finished washing clothes.

The transitions between these states are triggered by events, such as the user turning on the washing machine, selecting a cycle, or pressing start



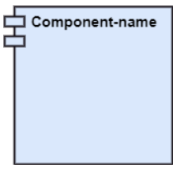
9. Explain about Component diagrams.

A component diagram is used to break down a large object-oriented system into the smaller components, so as to make them more manageable. It models the physical view of a system such as executables, files, libraries, etc. that resides within the node.

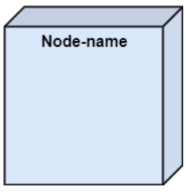
It visualizes the relationships as well as the organization between the components present in the system. It helps in forming an executable system. A component is a single unit of the system, which is replaceable and executable. The implementation details of a component are hidden, and it necessitates an interface to execute a function. It is like a black box whose behavior is explained by the provided and required interfaces.

Notation of a Component Diagram

a) A component



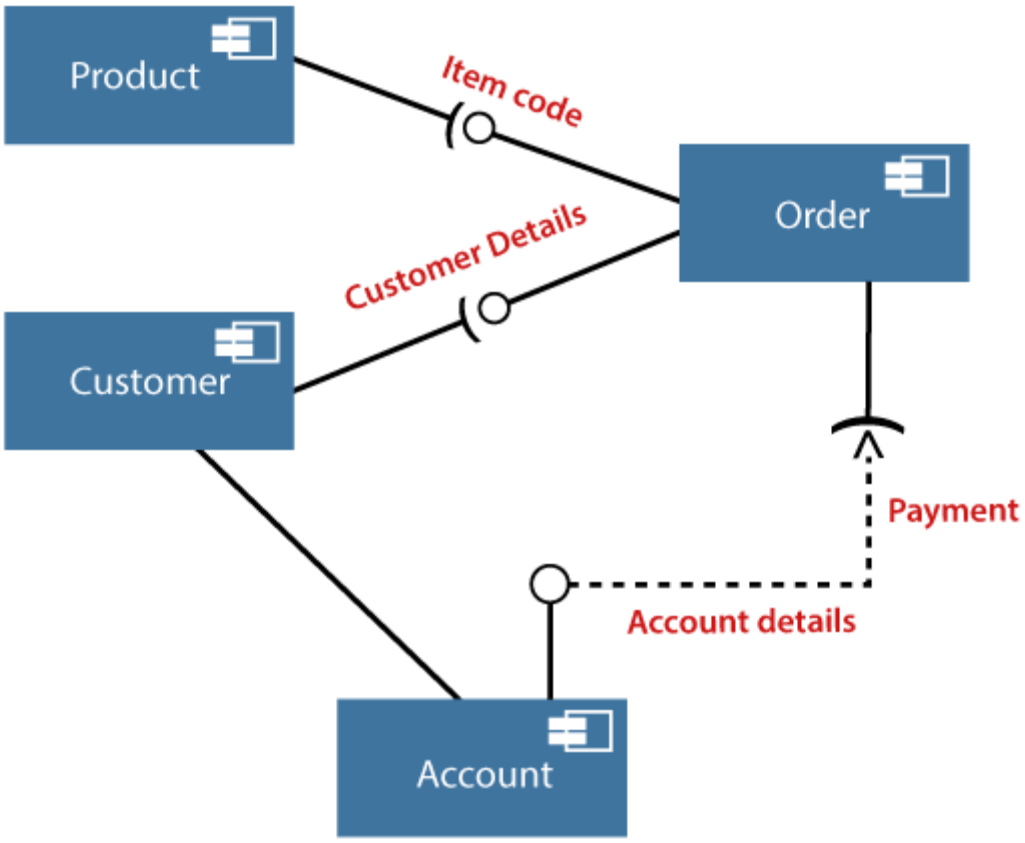
b) A node



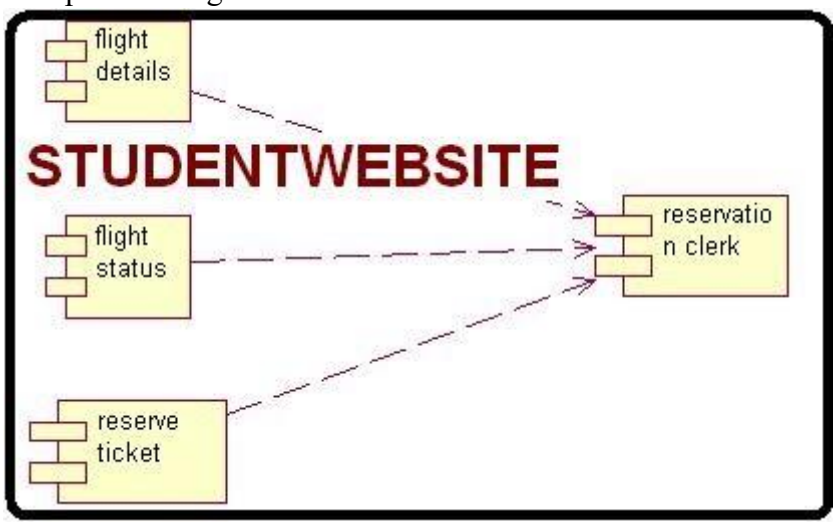
The component diagram also describes the static view of a system, which includes the organization of components at a particular instant. The collection of component diagrams represents a whole system.

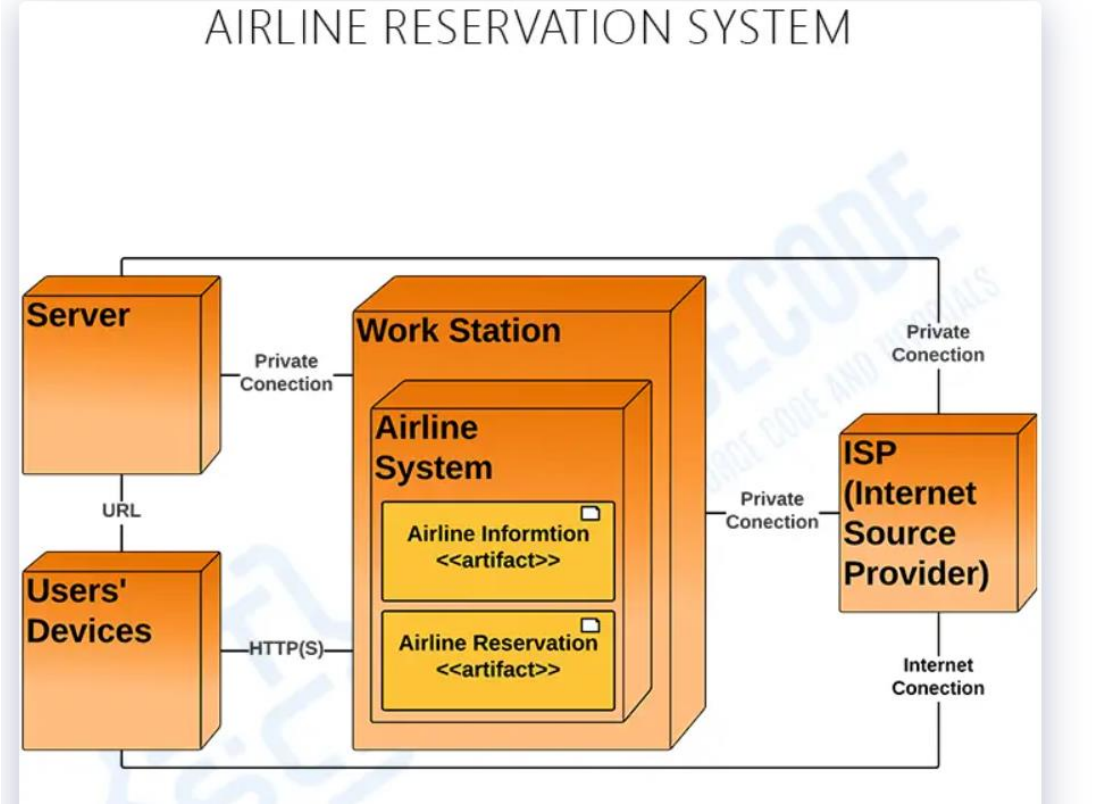
The main purpose of the component diagram are enlisted below:

1. It envisions each component of a system.
2. It constructs the executable by incorporating forward and reverse engineering.
3. It depicts the relationships and organization of components.



10. Draw Component and Deployment diagrams for Airline Booking.
    Component Diagram



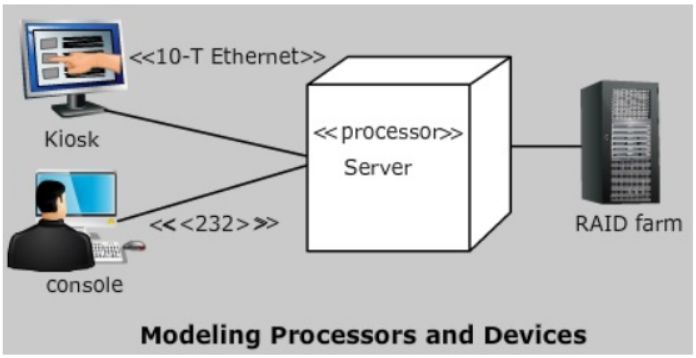


11. Architectural Modelling with Deployment diagrams.

Modeling Processors and Devices

To model processors and devices,

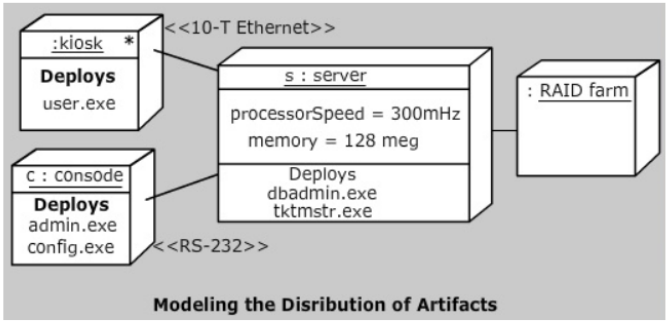
- Identify the computational elements of the system's deployment view and model each as a node
- If these elements represent generic processors and devices, then stereotype them as such,
- If they are kinds of processors and devices that are part of the vocabulary of your domain, then specify an appropriate stereotype with an icon for each
- As with class modeling, consider the attributes and operations that might apply to each node



Modeling the Distribution of Components

To model the distribution of artifacts,

- For each significant artifact in your system, allocate it to a given node
- Consider duplicate locations for components
- Some components (such as specific executable's and libraries) may reside on multiple nodes simultaneously



- Illustrate this allocation in one of the three ways:
- Use dependency relationships, connect each node with the artifacts it deploys
- List the component deployed on a node in an additional compartment Mention it in each node's specification which may not be shown explicitly

12. Forward and reverse engineering.

13. Difference between Forward Engineering and Reverse Engineering

Forward Engineering	Reverse Engineering
In forward engineering, the application are developed with the given requirements.	In reverse engineering or backward engineering, the information are collected from the given application.
Forward Engineering is a high proficiency skill.	Reverse Engineering or backward engineering is a low proficiency skill.
Forward Engineering takes more time to develop an application.	While Reverse Engineering or backward engineering takes less time to develop an application.
The nature of forward engineering is Prescriptive.	The nature of reverse engineering or backward engineering is Adaptive.
In forward engineering, production is started with given requirements.	In reverse engineering, production is started by taking the existing products.

Forward Engineering	Reverse Engineering
The example of forward engineering is the construction of electronic kit, construction of DC MOTOR , etc.	An example of backward engineering is research on Instruments etc.
Forward engineering Starts with requirements analysis and design, then proceeds to implementation and testing.	Reverse engineering Starts with an existing software system and works backward to understand its structure, design, and requirements.
Forward engineering is used to create new software applications from scratch.	Reverse engineering is Used to modify and improve an existing software application.
Forward egineering is process of moving from a high-level abstraction to a detailed implementation.	Reverse engineering is a process of moving from a low-level implementation to a higher-level abstraction.
Requires a clear set of requirements and design specifications.	Requirements and design specifications may not be available, making it necessary to reconstruct them from the code itself.
Forward engineering is generally more time-consuming and expensive.	Reverse engineering is generally less time-consuming and less expensive.
The final product is completely new and independent of any existing software system.	The final product is typically a modified or improved version of an existing software system.
Involves a series of steps such as requirements gathering, design, implementation, testing, and deployment.	Involves steps such as code analysis, code understanding, design recovery, and documentation.
Forward engineering is commonly used in the initial stages of software development.	Reverse engineering is commonly used in the maintenance stage of the software development life cycle.