Unit 3

State Machine View

State machine diagrams are also called as state chart diagrams. State machine diagrams are used to capture the behavior of a software system. UML State machine diagrams can be used to model the behavior of a class, a subsystem, a package, or even an entire system. It is also called a State chart or State Transition diagram.

State chart diagrams provide us an efficient way to model the interactions or communication that occurs within the external entities and a system. These diagrams are used to model the event-based system. A state of an object is controlled with the help of an event.

State chart diagrams are used to describe various states of an entity within the application system.

There are a total of two types of state machine diagrams:

1. **Behavioral state machine**

- It captures the behavior of an entity present in the system.
- It is used to represent the specific implementation of an element.
- The behavior of a system can be modelled using behavioral state machine diagrams.

2. **Protocol state machine**

- These diagrams are used to capture the behavior of a protocol.
- It represents how the state of protocol changes concerning the event. It also represents corresponding changes in the system.
- They do not represent the specific implementation of an element.

Why State Machine Diagram?

State chart diagram is used to capture the dynamic aspect of a system. State machine diagrams are used to represent the behavior of an application. An object goes through various states during its lifespan. The lifespan of an object remains until the program is terminated. The object goes from multiple states depending upon the event that occurs within the object. Each state represents some unique information about the object.

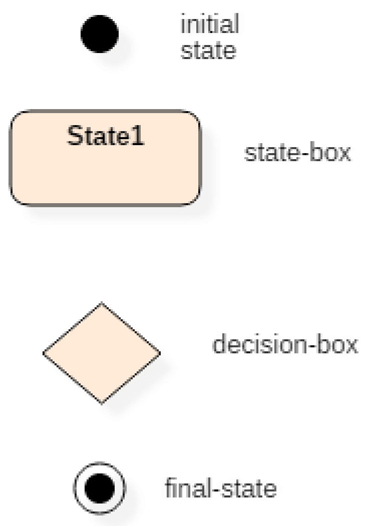
State chart diagrams are used to design interactive systems that respond to either internal or external event. State chart diagram visualizes the flow of execution from one state to another state of an object.

It represents the state of an object from the creation of an object until the object is destroyed or terminated.

The primary purpose of a state chart diagram is to model interactive systems and define each and every state of an object. State chart diagrams are designed to capture the dynamic behavior of an application system. These diagrams are used to represent various states of a system and entities within the system.

Notation and Symbol for State Machine

Following are the various notations that are used throughout the state chart diagram. All these notations, when combined, make up a single diagram.



UML state diagram notations

Initial state

The initial state symbol is used to indicate the beginning of a state machine diagram.

Final state

This symbol is used to indicate the end of a state machine diagram.

Decision box

It contains a condition. Depending upon the result of an evaluated guard condition, a new path is taken for program execution.

Transition

A transition is a change in one state into another state which is occurred because of some event. A transition causes a change in the state of an object.

State box

It is a specific moment in the lifespan of an object. It is defined using some condition or a statement within the classifier body. It is used to represent any static as well as dynamic situations.

It is denoted using a rectangle with round corners. The name of a state is written inside the rounded rectangle.

The name of a state can also be placed outside the rectangle. This can be done in case of composite or submachine states. One can either place the name of a state within the rectangle or outside the rectangle in a tabular box. One cannot perform both at the same time.

A state can be either active or inactive. When a state is in the working mode, it is active, as soon as it stops executing and transits into another state, the previous state becomes inactive, and the current state becomes active.

Types of State

Unified Modeling Language defines three types of states:

- Simple state
    - They do not have any substrate.

  Simple state may have 3 compartments

  1) Name Compartment: Holds the name of the state as a string. States w/o name are called anonymous states.

  2) Internal activities: Holds the list of internal action or states activities that are performed while the elements are in the state. The activity labeld identifies the circumstances under which the behavior specified by the activity expression will be involved.

  Several labels are reserved for special purpose and can not be used as event name.

  i) Entry (Behavior performed upon entry to the state)

  ii) Do (Performed as long as the elements is in the state)

iii) Exit (Behavior performed upon exit from the state)

3) Internal Transition: Contains a list of internal transition. Where each item has the form as described for trigger.

- Composite state
  - These types of states can have one or more than one substrate.
  - A composite state with two or more substrates (nested states) is called an orthogonal state.

Composite state has 4 compartments

i) Name ii) Internal activities iii) Internal Transition

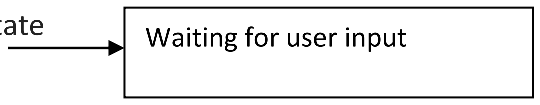
iv) Decomposition:

- Submachine state
  - These states are semantically equal to the composite states.
  - Unlike the composite state, we can reuse the submachine states.

- **Pseudo state :** It is used to connect multiple transition into more complex state transition paths.

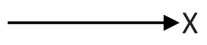
  Pseudo state includes :
  1) Initial : It represent a default vertex that is the source for a single transition of the default state.

  Intial Pseudo state

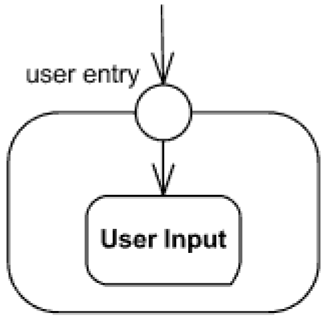


2) Terminate Pseudo state: It implies that the execution of this state machine by means of its context object is terminated.

Terminated Pseudo state is shown

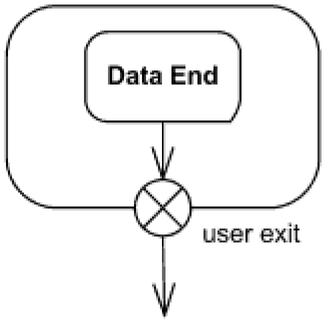


3) Entry Point: Entry point pseudo state is an entry point of a state machine or composite state. In each region of the state machine or composite state it has at most a single transition to a vertex within the same region.
An entry point is shown as a small circle on the border of the state machine diagram or composite state, with the name associated with it.



Entry point ***user entry***
Optionally it may be placed both within the state machine diagram and outside the border of the state machine diagram

4) Exit Point : Exit point pseudostate is an exit point of a state machine. Entering an exit point within any region of the state machine implies the exit of this composite state and the triggering of the transition that has this exit point as source in the state machine
An exit point is shown as a small circle with a cross on the border of the state machine diagram, with the name associated with it.



Exit point ***user exit***

How to draw a State chart diagram?

State chart diagrams are used to describe the various state that an object passes through. A transition between one state into another state occurs because of some triggered event. To draw a state diagram, one must identify all the possible states of any particular entity.

The purpose of these UML diagrams is to represent states of a system. States plays a vital role in state transition diagrams. All the essential object, states, and the events that cause changes within the states must be analyzed first before implementing the diagram.

Following rules must be considered while drawing a state chart diagram:

1. The name of a state transition must be unique.
2. The name of a state must be easily understandable and describe the behavior of a state.
3. If there are multiple objects, then only essential objects should be implemented.
4. Proper names for each transition and an event must be given.

When to use State Diagrams?

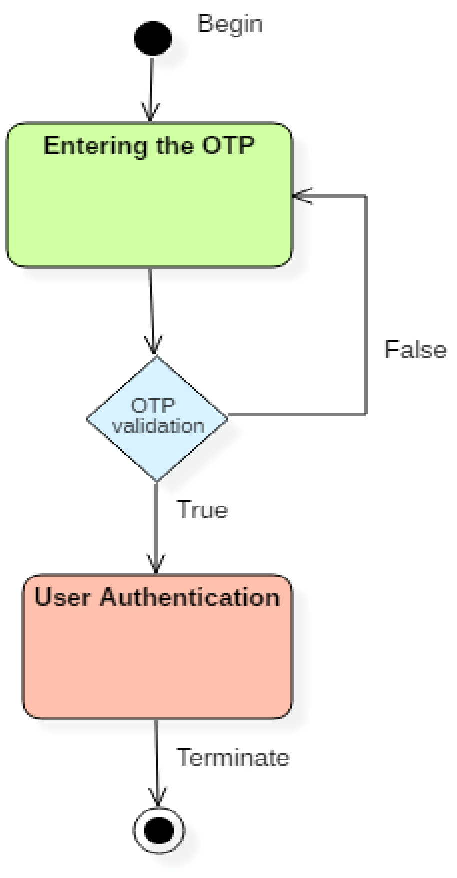
State diagrams are used to implement real-life working models and object-oriented systems in depth. These diagrams are used to compare the dynamic and static nature of a system by capturing the dynamic behavior of a system.

State chart diagrams are used to capture the changes in various entities of the system from start to end. They are used to analyze how an event can trigger change within multiple states of a system.

State char diagrams are used,

1. To model objects of a system.
2. To model and implement interactive systems.
3. To display events that trigger changes within the states.

Example:



State machine vs. Flowchart

State machine	Flow Chart
It represents various states of a system.	The Flowchart illustrates the program execution flow.
The state machine has a WAIT concept, i.e., wait for an action or an event.	The Flowchart does not deal with waiting for a concept.
State machines are used for a live running system.	Flowchart visualizes branching sequences of a system.
The state machine is a modeling diagram.	A flowchart is a sequence flow or a DFD diagram.
The state machine can explore various states of a system.	Flowchart deal with paths and control flow.

Activity Diagram:

Activity diagram is defined as a UML diagram that focuses on the execution and flow of the behavior of a system instead of implementation. It is also called **object-oriented flowchart**. Activity diagrams consist of activities that are made up of actions which apply to behavioral modeling technology.

Components of Activity Diagram

Activities

It is a behavior that is divided into one or more actions. Activities are a network of nodes connected by edges. There can be action nodes, control nodes, or object nodes. Action nodes represent some action. Control nodes represent the control flow of an activity. Object nodes are used to describe objects used inside an activity. Edges are used to show a path or a flow of execution. Activities start at an initial node and terminate at a final node.

Activity partition/swim lane

An activity partition or a swim lane is a high-level grouping of a set of related actions. A single partition can refer to many things, such as classes, use cases, components, or interfaces.

If a partition cannot be shown clearly, then the name of a partition is written on top of the name of an activity.

Fork and Join nodes

Using a fork and join nodes, concurrent flows within an activity can be generated. A fork node has one incoming edge and numerous outgoing edges. It is similar to one too many decision parameters. When data arrives at an incoming edge, it is duplicated and split across numerous outgoing edges simultaneously. A single incoming flow is divided into multiple parallel flows.

A join node is opposite of a fork node as It has many incoming edges and a single outgoing edge. It performs logical AND operation on all the incoming edges. This helps you to synchronize the input flow across a single output edge.

Why use Activity Diagrams?

Activity diagram allows you to create an event as an activity which contains a collection of nodes joined by edges. An activity can be attached to any modeling element to model its behavior. Activity diagrams are used to model,

- Use cases
- Classes
- Interfaces
- Components
- Collaborations

  Activity diagrams are used to model processes and workflows. The essence of a useful activity diagram is focused on communicating a specific aspect of a system's dynamic behavior. Activity diagrams capture the dynamic elements of a system.

  Activity diagram is similar to a flowchart that visualizes flow from one activity to another activity. Activity diagram is identical to the flowchart, but it is not a flowchart. The flow of activity can be controlled using various control elements in the UML diagram. In simple words, an activity diagram is used to activity diagrams that describe the flow of execution between multiple activities.

Activity Diagram Notations

Activity diagrams symbol can be generated by using the following notations:

- Initial states: The starting stage before an activity takes place is depicted as the initial state
- Final states: The state which the system reaches when a specific process ends is known as a Final State
- State or an activity box:
- Decision box: It is a diamond shape box which represents a decision with alternate paths. It represents the flow of control.

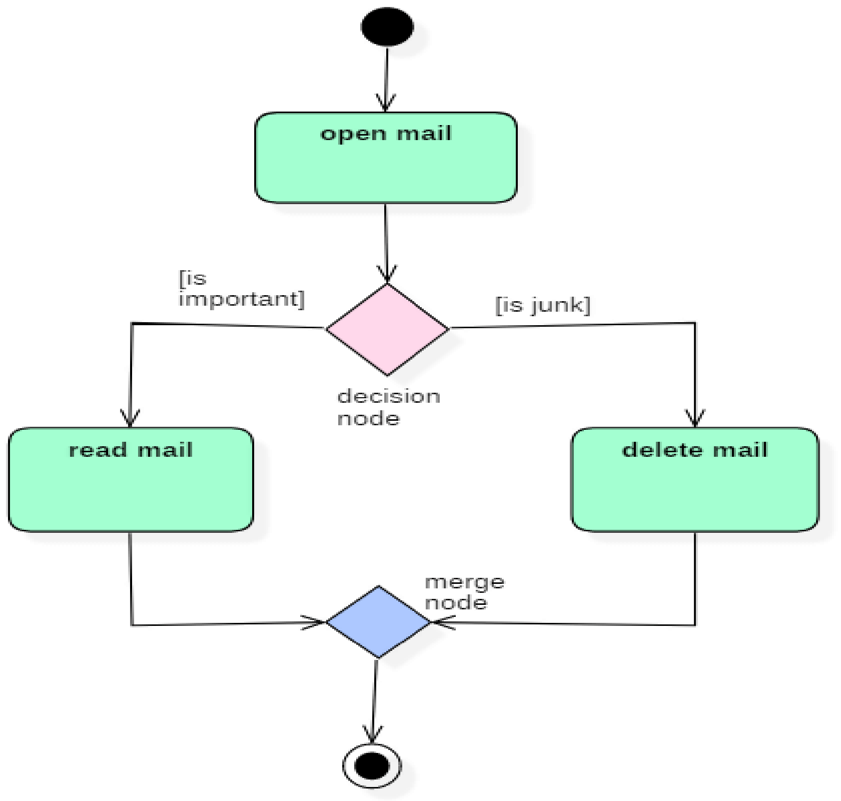
How to draw an activity diagram?

- Activity diagram is a flowchart of activities. It represents the workflow between various system activities. Activity diagrams are similar to the flowcharts, but they are not flowcharts. Activity diagram is an advancement of a flowchart that contains some unique capabilities.
- Activity diagrams include swim lanes, branching, parallel flow, control nodes, expansion nodes, and object nodes. Activity diagram also supports exception handling.
- To draw an activity diagram, one must understand and explore the entire system. All the elements and entities that are going to be used inside the diagram must be known by the user. The central concept which is nothing but an activity must be clear to the user. After analyzing all activities, these activities should be explored to find various constraints that are applied to activities. If there is such a constraint, then it should be noted before developing an activity diagram.

Following rules must be followed while developing an activity diagram,

1. All activities in the system should be named.
2. Activity names should be meaningful.
3. Constraints must be identified.
4. Activity associations must be known.

Example: In the below activity diagram, three activities are specified. When the mail checking process begins user checks if mail is important or junk. Two guard conditions [is essential] and [is junk] decides the flow of execution of a process. After performing the activity, finally, the process is terminated at termination node.



When Use Activity Diagram

Activity diagram is used to model business processes and workflows. These diagrams are used in software modeling as well as business modeling.

Most commonly activity diagrams are used to,

1. Model the workflow in a graphical way, which is easily understandable.
2. Model the execution flow between various entities of a system.
3. Model the detailed information about any function or an algorithm which is used inside the system.
4. Model business processes and their workflows.
5. Capture the dynamic behavior of a system.
6. Generate high-level flowcharts to represent the workflow of any application.
7. Model high-level view of an object-oriented or a distributed system

<u>Interaction Diagram</u>:

An interaction is defined as a behavior that consists of a group of messages exchanged among elements of accomplishing a specific task in the system. An interaction may use any of the features of its classifier context. It can use any of the classifier contexts which it has access. The critical component in an interaction diagram is lifeline and messages.

Interaction overview diagrams are provided by UML to establish communication between objects. It does not manipulate the data associated with the particular communication path. Interaction diagrams mostly focus on message passing and how these messages make up one functionality of a system. Interaction diagrams are designed to display how the objects will realize the particular requirements of a system.

Various UML elements typically own interaction diagrams. The details of interaction can be shown using several notations such as sequence diagram, timing diagram, and communication/collaboration diagram.

Interaction diagrams capture the dynamic behavior of any system.

Following are the different types of interaction diagrams defined in UML:

- Sequence diagram
- Collaboration diagram
- Timing diagram

The purpose of a sequence diagram in UML is to visualize the sequence of a message flow in the system. The sequence diagram shows the interaction between two lifelines as a time-ordered sequence of events.

The collaboration diagram is also called as a communication diagram. The purpose of a collaboration diagram is to emphasize structural aspects of a system, i.e., how various lifelines in the system connect.

Timing diagrams focus on the instance at which a message is sent from one object to another object.

Purpose of an Interaction Diagram

Interaction diagrams help you to visualize the interactive behavior of a system. Interaction diagrams are used to represent how one or more objects in the system connect and communicate with each other.

Interaction diagrams focus on the dynamic behavior of a system. An interaction diagram provides us the context of an interaction between one or more lifelines in the system.

In UML, the interaction diagrams are used for the following purposes:

- Interaction diagrams are used to observe the dynamic behavior of a system.
- Interaction diagram visualizes the communication and sequence of message passing in the system.
- Interaction diagram represents the structural aspects of various objects in the system.
- Interaction diagram represents the ordered sequence of interactions within a system.
- Interaction diagram provides the means of visualizing the real time data via UML.

Important terminology

An interaction diagram contains lifelines, messages, operators, state invariants and constraints.

Lifeline

A lifeline represents a single participant in an interaction. It describes how an instance of a specific classifier participates in the interaction.

A lifeline represents a role that an instance of the classifier may play in the interaction. Following are various attributes of a lifeline,

1. Name
    1. It is used to refer the lifeline within a specific interaction.
    2. A name of a lifeline is optional.
2. Type
    1. It is the name of a classifier of which the lifeline represents an instance.
3. Selector
    1. It is a Boolean condition which is used to select a particular instance that satisfies the requirement.
    2. Selector attribute is also optional.

The notation of lifeline is explained in the notation section.

Messages

A message is a specific type of communication between two lifelines in an interaction. A message involves following activities,

1. A call message which is used to call an operation.
2. A message to create an instance.
3. A message to destroy an instance.
4. For sending a signal.

When a lifeline receives a call message, it acts as a request to invoke an operation that has a similar signature as specified in the message. When a lifeline is executing a message, it has a focus of control. As the interaction progresses over time, the focus of control moves between various lifelines. This movement is called a flow of control.

Message Name	Meaning
Synchronous message	The sender of a message keeps waiting for the receiver to return control from the message execution.
Asynchronous message	The sender does not wait for a return from the receiver; instead, it continues the execution of a next message.
Return message	The receiver of an earlier message returns the focus of control to the sender.
Object creation	The sender creates an instance of a classifier.
Object destruction	The sender destroys the created instance.
Found message	The sender of the message is outside the scope of interaction.
Lost message	The message never reaches the destination, and it is lost in the interaction.

Operator

An operator specifies an operation on how the operands are going to be executed. The operators in UML support operations on data in the form of branching as well as iteration. Various operators can be used to ensure the use of iteration and branching in the UML model.

Operator	Name	Meaning
Opt	Option	An operand is executed if the condition is true. e.g., If else
Alt	Alternative	The operand, whose condition is true, is executed. e.g., switch
Loop	Loop	It is used to loop an instruction for a specified period.
Break	Break	It breaks the loop if a condition is true or false, and the next instruction is executed.
Ref	Reference	It is used to refer to another interaction.
Par	Parallel	All operands are executed in parallel.

Iteration

In an interaction diagram, we can also show iteration using an iteration expression. An iteration expression consists of an iteration specifier and an optional iteration clause. There is no pre-specified syntax for UML iteration.

Branching

In an interaction diagram, we can represent branching by adding guard conditions to the messages. Guard conditions are used to check if a message can be sent forward or not. A message is sent forward only when its guard condition is true. A message can have multiple guard conditions, or multiple messages can have the same guard condition. Branching in UML is achieved with the help of alt and opt, operators.

How to draw a Interaction diagram?

Interaction diagrams are used to represent the interactive behavior of a system. Interaction diagrams focus on the dynamic behavior of a system. An interaction diagram provides us the context of an interaction between one or more lifelines in the system.

To draw an interaction diagram, you have first to determine the scenario for which you have to draw an interaction diagram. After deciding the situation, identify various lifelines that are going to be involved in the interaction. Categorize all the lifeline elements and explore them to identify possible connections and how the lifelines are related to one another. To draw an interaction diagram, the following things are required:

1. The total number of lifelines that are going to be part of an interaction
2. is a sequence of message flow within various objects of a system.
3. Various operators to ease the functionality of an interaction diagram.
4. Various types of messages to display the interaction more clearly and in a precise manner.
5. The ordered sequence of messages.
6. Organization and a structure of an object.
7. Various time constructs of an object.

Use of an interaction diagram

Interaction diagrams consist of a sequence diagram, collaboration diagram, and timing diagrams. Following is the specific purpose of an interaction diagram:

- Sequence diagrams are used to explore any real application or a system.
- Interaction diagrams are used to explore and compare the use of sequence, collaborations, and timing diagrams.
- Interaction diagrams are used to capture the behavior of a system. It displays the dynamic structure of a system.
- Sequence diagrams are used to represent message flow from one object to another object.
- Collaboration diagrams are used to understand the object architecture of a system rather than message flow.
- Interaction diagrams are used to model a system as a time-ordered sequence of events.
- Interaction diagrams are used in reverse as well as forward engineering.
- Interaction diagrams are used to organize the structure of interactive elements.

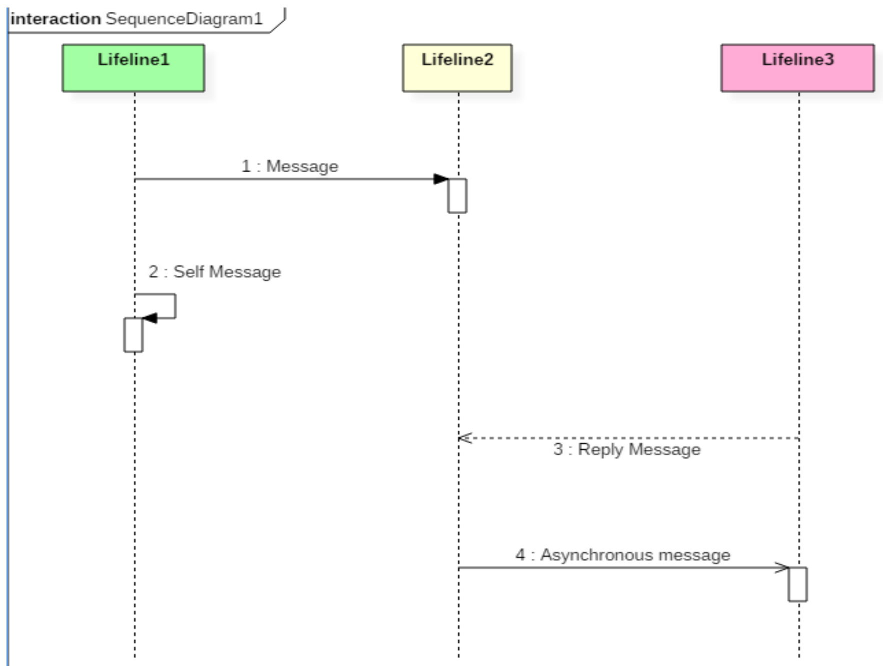
Types of Interaction diagram and Notations

Following are the different types of interaction diagrams defined in UML:

- Sequence diagram
- Collaboration diagram
- Timing diagram

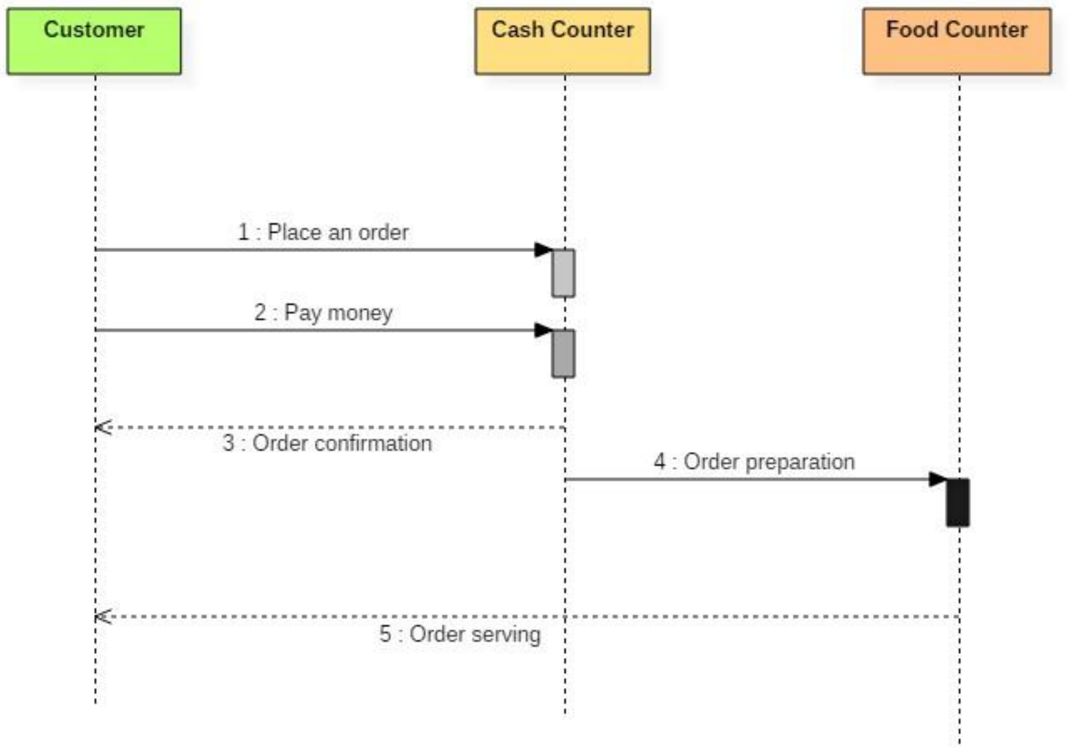
Sequence Diagram

- The purpose of a sequence diagram in UML is to visualize the sequence of a message flow in the system.
- The sequence diagram shows the interaction between two lifelines as a time-ordered sequence of events.
- A sequence diagram is used to capture the behavior of any scenario.
- A sequence diagram shows an implementation of a scenario in the system. Lifelines in the system take part during the execution of a system.
- In a sequence diagram, a lifeline is represented by a vertical bar.
- A message flow between two or more objects is represented using a vertical dotted line which extends across the bottom of the page.
- In a sequence diagram, different types of messages and operators are used which are described above.
- In a sequence diagram, iteration and branching are also used.



The above sequence diagram contains lifeline notations and notation of various messages used in a sequence diagram such as a create, reply, asynchronous message, etc.

Example :



Sequence diagram of ordering system

The ordered sequence of events in a given sequence diagram is as follows:

1. Place an order.
2. Pay money to the cash counter.
3. Order Confirmation.
4. Order preparation.
5. Order serving.

If one changes the order of the operations, then it may result in crashing the program. It can also lead to generating incorrect or buggy results. Each sequence in the above-given sequence diagram is denoted using a different type of message. One cannot use the same type of message to denote all the interactions in the diagram because it creates complications in the system.

You must be careful while selecting the notation of a message for any particular interaction. The notation must match with the particular sequence inside the diagram.

Benefits of a Sequence Diagram

- Sequence diagrams are used to explore any real application or a system.
- Sequence diagrams are used to represent message flow from one object to another object.
- Sequence diagrams are easier to maintain.
- Sequence diagrams are easier to generate.
- Sequence diagrams can be easily updated according to the changes within a system.
- Sequence diagram allows reverse as well as forward engineering.

Drawbacks of a sequence diagram

- Sequence diagrams can become complex when too many lifelines are involved in the system.
- If the order of message sequence is changed, then incorrect results are produced.
- Each sequence needs to be represented using different message notation, which can be a little complex.
- The type of message decides the type of sequence inside the diagram.

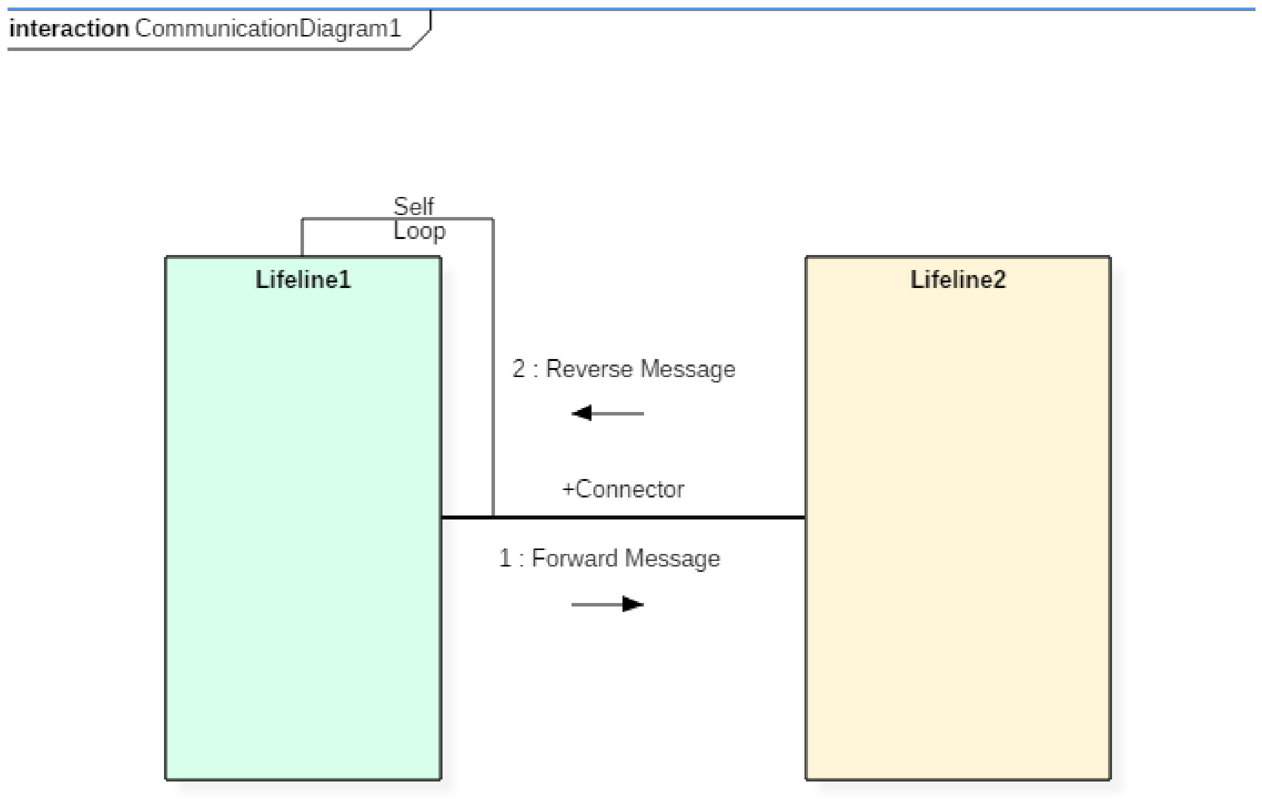
What is the Collaboration diagram?

A collaboration diagram is required to identify how various objects make up the entire system. They are used to understand the object architecture within a system rather than the flow of a message in a sequence diagram. An object an entity that has various attributes associated with it. It is a concept of object-oriented programming. There are multiple objects present inside an object-oriented system where each object can be associated with any other object inside the system. Collaboration or communication diagrams are used to explore the architecture of objects inside the system. The message flow between the objects can be represented using a collaboration diagram.

Benefits of Collaboration Diagram

- It is also called as a communication diagram.
- It emphasizes the structural aspects of an interaction diagram - how lifeline connects.
- Its syntax is similar to that of sequence diagram except that lifeline don't have tails.
- Messages passed over sequencing is indicated by numbering each message hierarchically.
- Compared to the sequence diagram communication diagram is semantically weak.
- Object diagrams are special case of communication diagram.

- It allows you to focus on the elements rather than focusing on the message flow as described in the sequence diagram.
- Sequence diagrams can be easily converted into a collaboration diagram as collaboration diagrams are not very expressive.
- While modeling collaboration diagrams w.r.t sequence diagrams, some information may be lost.

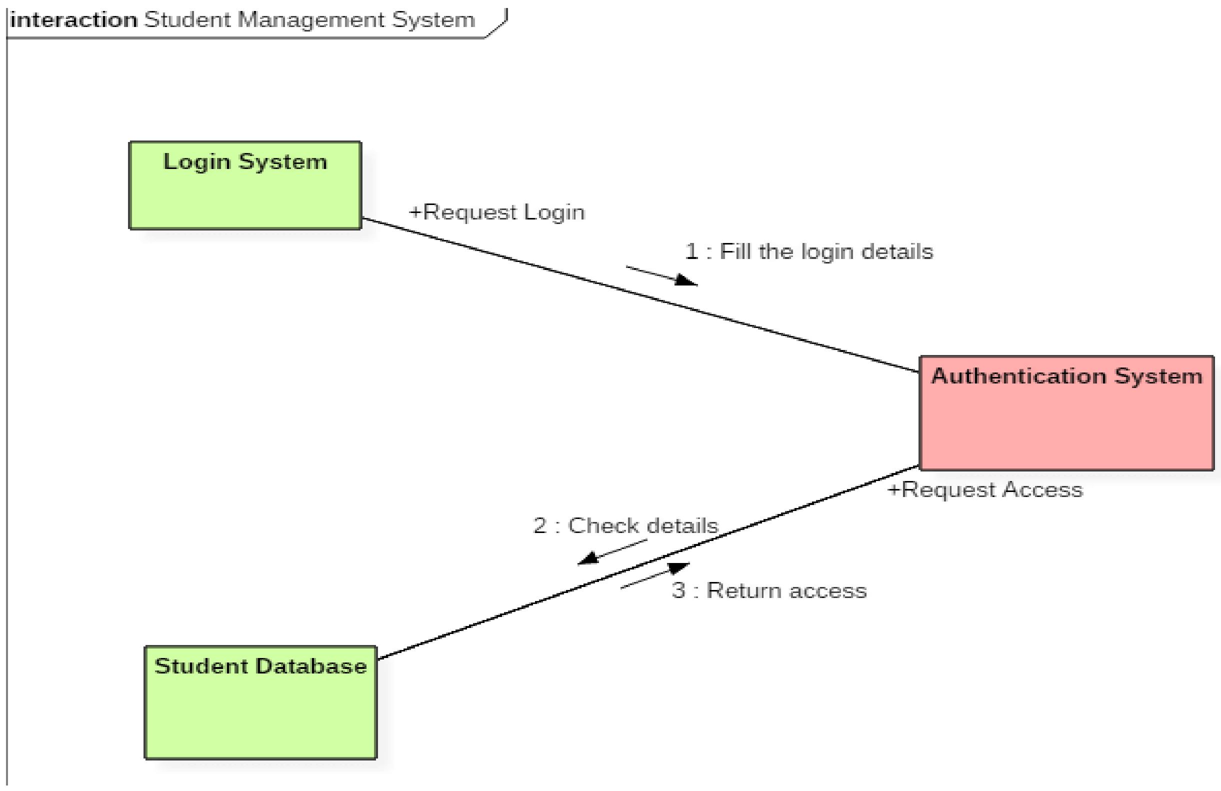


Drawbacks of a Collaboration Diagram

- Collaboration diagrams can become complex when too many objects are present within the system.
- It is hard to explore each object inside the system.
- Collaboration diagrams are time consuming.
- The object is destroyed after the termination of a program.
- The state of an object changes momentarily, which makes it difficult to keep track of every single change the occurs within an object of a system

Collaboration diagram Example

Following diagram represents the sequencing over student management system:



Collaboration diagram for student management system

The above collaboration diagram represents a student information management system. The flow of communication in the above diagram is given by,

1. A student requests a login through the login system.
2. An authentication mechanism of software checks the request.
3. If a student entry exists in the database, then the access is allowed; otherwise, an error is returned.