

# CHAPTER 1: UML DIAGRAMS

---

## 1.6 INTERACTION DIAGRAMS

- ❑ Interaction diagrams describe how groups of objects collaborate in some behavior.
- ❑ The UML includes **interaction diagrams** to illustrate how objects interact via messages.
- ❑ They are used for dynamic object modelling.
- ❑ The two main types of interaction diagrams are :
  - Sequence diagrams
  - Communication diagram

### **Interaction view of the system:**

- ★ An **interaction** is a set of messages within collaboration that are exchanged by **classifier roles** across **association roles**.
- ★ When collaboration exists at run time, **objects** bound to classifier roles exchange **message** instances across **links** bound to association roles.
- ★ An interaction models the execution of an operation, use case, or other behavioral entity
- ★ A **message** is a one-way communication between two objects, a flow of control with information from a **sender** to a **receiver**.
- ★ A message may have **parameters** that convey **values** between the objects. A message can also be a **call** (the synchronous invocation of an operation with a mechanism for later returning control to the sender).

### **Purpose of Interaction Diagrams:**

The purposes of interaction diagrams are to visualize the interactive behaviour of the system.

So the purposes of interaction diagram can be describes as:

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

### Differences between sequence and collaboration diagram:

Sequence Diagrams	Communication diagrams
Sequence diagrams show the interaction by showing each participant with a lifeline that runs vertically down the page and the ordering of messages by reading down the page.	Collaboration diagram emphasizes on the structural organization of the objects that send and receive messages.
Illustrate interaction in a <b>fence format</b> ,	Illustrates object interaction in a <b>graph or network format</b> .
Additional objects in a sequence diagrams must extend to the right, which is limiting.	collaboration diagrams have the advantage of allowing vertical expansion for new objects;
Clearly shows sequence and time ordering of messages	Collaboration diagram examples make it harder to easily see the sequence of messages.

## SEQUENCE DIAGRAMS:

Typically, a sequence diagram captures the behavior of a single scenario. The diagram shows a number of example objects and the messages that are passed between these objects within the use case. They are used in conjunction with the class diagrams to capture most of the information about a system.

### Elements of Sequence diagrams:

1. The vertical dimension is the **time axis**; time proceeds down the page.
2. The horizontal dimension shows the **classifier roles** that represent individual **objects** in the collaboration.
3. **Lifeline**: Each classifier role is represented by a vertical column—the **lifeline**.
  - a. During the time an object exists, the role is shown by a dashed line.
  - b. Each lifeline has an activation bar that shows when the participant is active in the interaction.
  - c. This corresponds to one of the participant's methods being on the stack.
4. **Activation**: During the time an **activation** of a procedure on the object is active, the lifeline is drawn as a double line.
  - a. Activation is the execution of a procedure, including the time it waits for nested procedures to execute.
  - b. It is shown by a double line replacing part of the **lifeline** in a sequence diagram.

5. **Active Objects:** An **active object** is one that holds the root of a stack of activations. Each active object has its own event-driven thread of control that executes in parallel with other active objects
6. **Message:** A **message** is shown as an arrow from the lifeline of one object to that of another. The arrows are arranged in time sequence down the diagram.
7. **Frames:** Frames consist of some region of a sequence diagram that is divided into one or more fragments. Each frame has an Operator and each fragment may have a guard.
8. **Synchronous and asynchronous calls:** If a caller sends a synchronous message, it must wait until the message is done, such as invoking a subroutine. If a caller sends an asynchronous message, it can continue processing and doesn't have to wait for a response.



FIG: Basic Sequence Diagram Notation

#### Notations and symbols in Sequence diagram:

##### 1 . Links

- a. Unlike collaboration diagrams, sequence diagrams do not show links.

##### 2 . Messages

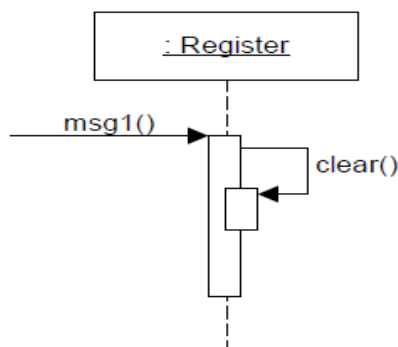
- a. Each message between objects is represented with a message expression on an arrowed line between the objects. The time ordering is organized from top to bottom.

##### 3 . Illustrating Returns

- a. A sequence diagram may optionally show the return from a message as a dashed open-arrowed line at the end of an activation box.

##### 4 . Messages to "self" or "this"

- a. A message can be illustrated as being sent from an object to itself by using a nested activation box



## 5 . Conditional Messages

A conditional message is shown in Figure 15.22.

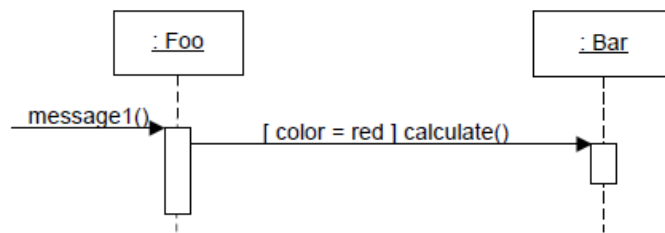


Figure 15.22 A conditional message.

## 6 . Mutually Exclusive Conditional Messages

### *Mutually Exclusive Conditional Messages*

The notation for this case is a kind of angled message line emerging from a common point, as illustrated in Figure 15.23.

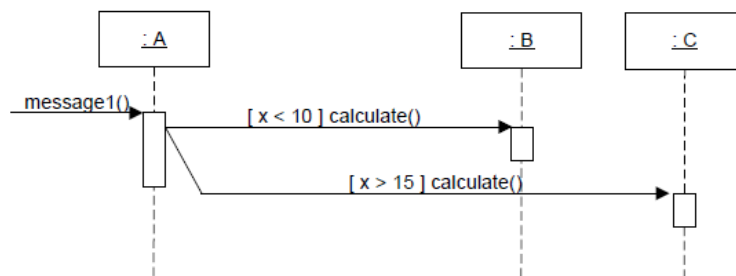


Figure 15.23 Mutually exclusive conditional messages.

## 7 . Iteration for a single Message

### *Iteration for a Single Message*

Iteration notation for one message is shown in Figure 15.24.

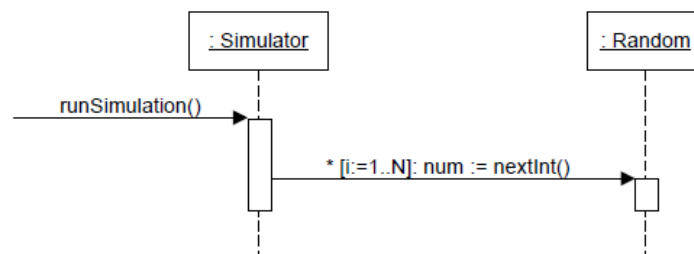


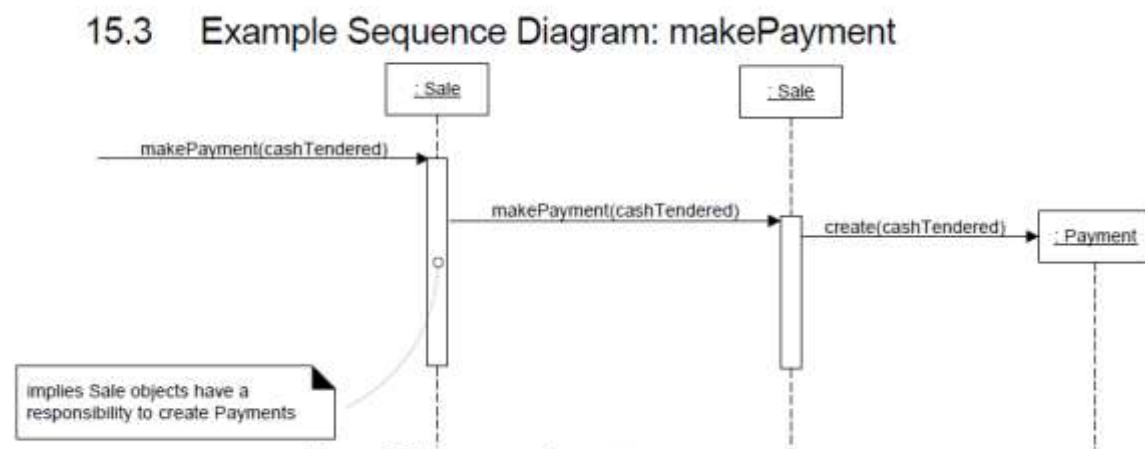
Figure 15.24 Iteration for one message.

## Steps in Creating Sequence diagrams:

1. Identify a set of classes involved in a scenario.
2. Create generic objects for these classes.
3. List the objects in a line along the top of a page and drop a dotted line beneath each object. These dotted lines are called lifelines.
4. Define who will initiate the interaction
5. Draw the first message to a sub-system -Specify the message sent from the actor who begins the interaction to the first point of contact in the system.
6. Draw message to other sub-systems-Send other messages between objects (i.e. lifelines) in the system
7. Draw return message to actor (if any) - Send return messages back to the original callers upon receiving their messages.

For the **NextGEN POS system**, consider the following scenario:

1. The message make Payment is sent to an instance of a Register. The sender is not identified.
2. The Register instance sends the make Payment message to a Sale instance.
3. The Sale instance creates an instance of a Payment.



## COLLABORATION DIAGRAM:

- Model collaborations between objects or roles that deliver the functionalities of use cases and operations.
- Model mechanisms within the architectural design of the system.
- Capture interactions that show the passed messages between objects and roles within the collaboration.

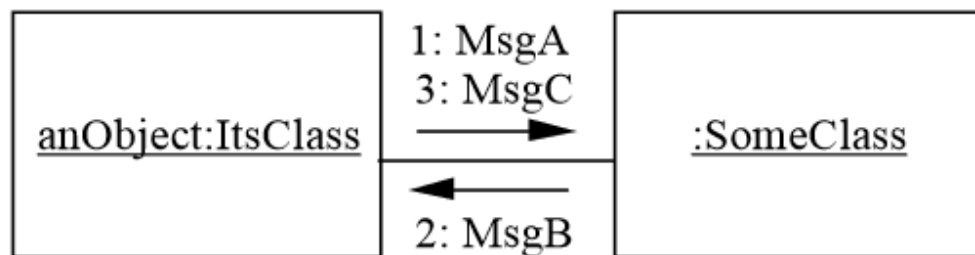
### Collaboration:

- ★ Collaboration is a collection of named objects and actors with links connecting them. They collaborate in performing some task.
- ★ Collaboration defines a set of participants and relationships that are meaningful for a given set of purposes.
- ★ Objects collaborate by communicating (passing messages) with one another in order to work together.

### Messages:

- The method calls are similar to that of a sequence diagram. But the difference is that the sequence diagram does not describe the object organization whereas the collaboration diagram shows the object organization.
- Communication is made up of messages
  - Messages are shown as labelled arrows
  - Numbers show sequence

## Notation



In collaboration diagram the method call sequence is indicated by some numbering technique. The number indicates how the methods are called one after another.

### Constructing Collaboration Diagrams

1. Identify behavior whose realization and implementation is specified
2. Identify the structural elements (class roles, objects, subsystems) necessary to carry out the functionality of the collaboration
  - Decide on the context of interaction: system, subsystem, use case and operation
3. Model structural relationships between those elements to produce a diagram showing the context of the interaction
4. Consider the alternative scenarios that may be required
  - Draw instance level collaboration diagrams, if required

Thus, for the NextGEN POS system, the collaboration diagram is drawn as follows:

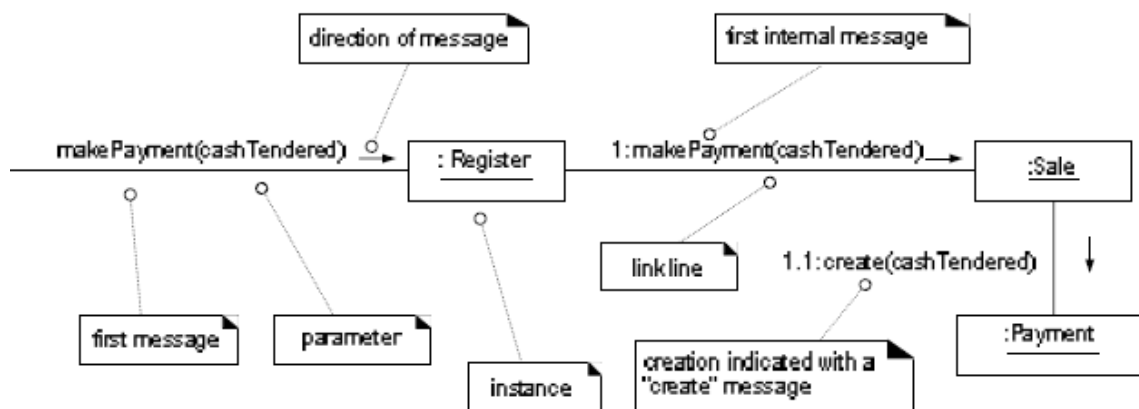


Figure 15.3 Collaboration diagram.

## Basic Collaboration diagram Notations:

### 1. Links

A **link** is a connection path between two objects; it indicates some form of navigation and visibility between the objects is possible. More formally, a link is an instance of an association. For example, there is a link or path of navigation from a *Register* to a *Sale*, along which messages may flow, such as the *makePayment* message

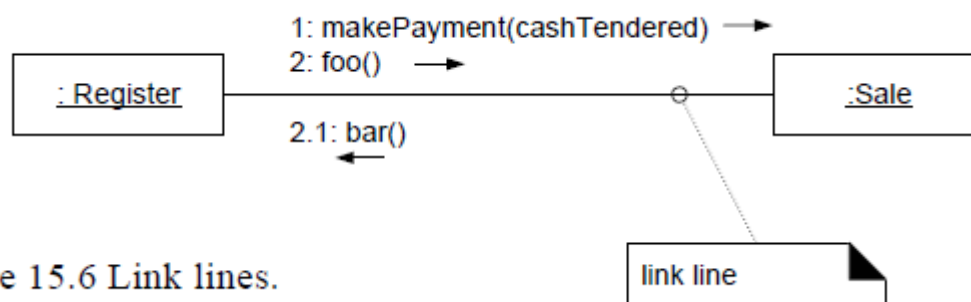


Figure 15.6 Link lines.

### 2. Self-Delegation

#### Messages to "self" or "this"

A message can be sent from an object to itself (Figure 15.8). This is illustrated by a link to itself, with messages flowing along the link.

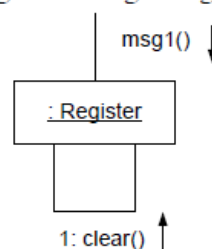


Figure 15.8 Messages to "this."

### 3. Creation of Instances

- ★ Any message can be used to create an instance, but there is a convention in the UML to use a message named *create* for this purpose.
- ★ The *create* message may include parameters, indicating the passing of initial values. This indicates, for example, a constructor call with parameters in Java.

highlight the creation.

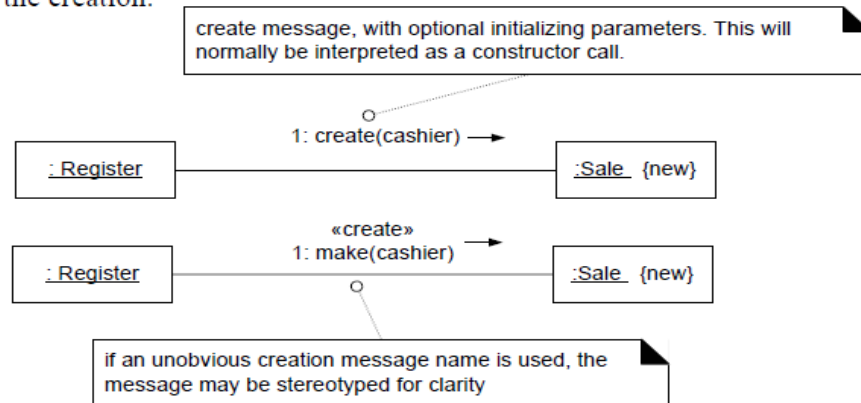


Figure 15.9 Instance creation.

## 5. Conditional Messages

### Conditional Messages

A conditional message (Figure 15.12) is shown by following a sequence number with a conditional clause in square brackets, similar to an iteration clause. The message is only sent if the clause evaluates to *true*.

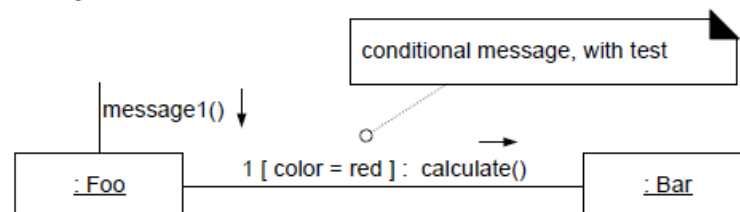


Figure 15.12 Conditional message.

## 6. Iteration or looping

### Iteration or Looping

Iteration notation is shown in Figure 15.14. If the details of the iteration clause are not important to the modeler, a simple '\*' can be used.

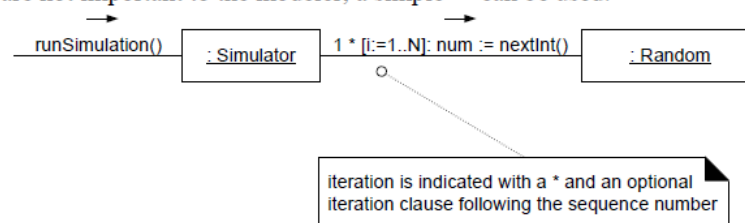


Figure 15.14 Iteration.



## 1.7 ACTIVITY DIAGRAMS

- ❑ UML Activity diagrams show **sequential and parallel activities** in a process.
- ❑ Activity diagram is basically a flow chart to represent the flow from one activity to another activity. The diagram can show both control flow and data flow.
- ❑ The activity can be described as an operation of the system. So the control flow is drawn from one operation to another.
- ❑ Activity diagrams deal with all type of flow control by using different elements like fork, join etc.
- ❑ Activity Diagrams are good for describing synchronization and concurrency between activities

### Need for activity diagrams:

- [1]. They are used for modelling
  1. Business processes
  2. Workflows
  3. Dataflows
  4. Complex algorithm
- [2]. Identify pre- and post-conditions for use cases

### Activity view of the system:



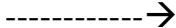



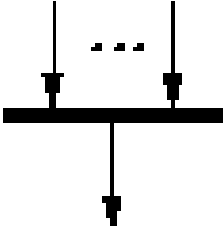
- a) An activity graph contains **activity states**.
- b) An activity state represents the execution of a statement in a procedure or the performance of an **activity** in a workflow.
- c) An activity diagram may contain **branches**, as well as **forking** of control into concurrent threads

### Swimlanes:

- It is often useful to organize the activities in a model according to responsibility.
- For example, by grouping together all the activities handled by one business organization.
- This kind of assignment can be shown by organizing the activities into distinct regions separated by lines in the diagram.
- Because of their appearance, each region is called a **swimlane**.

### Elements of activity diagram:

Activity Diagrams consist of activities, states and transitions between activities and states.

Symbol	Description
	<b>Activity:</b> is used to represent a set of actions.
	<b>Control Flow:</b> shows the sequence of execution
	<b>Data Flow:</b> shows the flow of an object from one activity to another activity.
	<b>Initial Node:</b> portrays the beginning of a set of actions or activities.
	<b>Final node:</b> denotes the end of workflow.
	<b>Decision node:</b> is used to represent a test condition to ensure that the control flow or object flow only goes down one path.
	<b>Join node:</b> is used to bring back together a set of parallel or concurrent flow of activities.

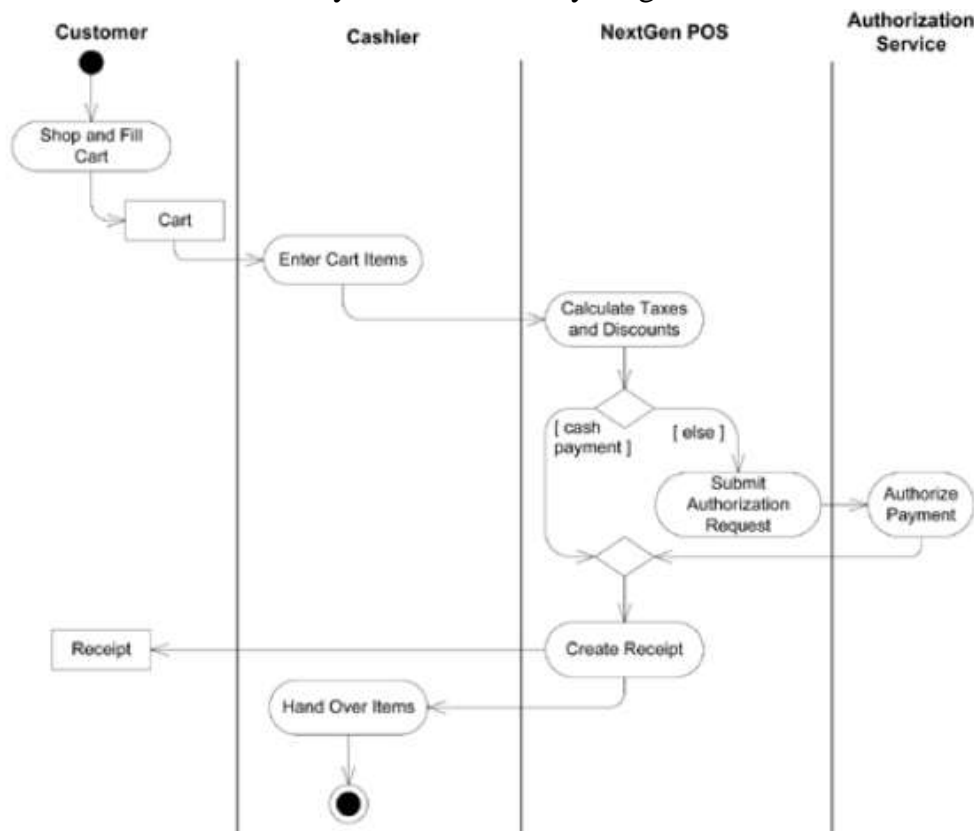
- 1) **Activity:** An Activity is the process being modelled. An activity is like a state where the criterion for leaving the state is the completion of the activity
- 2) **Transitions :** A Transition is the movement from one activity to another, the change from one state to another, or the movement between a state and an activity in either direction
  - i. They are unlabelled arrows from one activity to the next
  - ii. Transitions take place when one activity is complete and the next can commence
- 3) **Activity Edges:** The flow of an activity is shown using arrowed lines called edges or paths
- 4) **Control-flow:** Transitions indicate the order of action states
- 5) **Object- flow:** Transitions indicate that an action state inputs or outputs an object

- 6) **Forks:** A transition can be split into multiple paths and multiple paths combined into single transitions by using a synchronisation bar.
  - a. A fork is where the paths split
- 7) **Joins:** A join is where the paths meet. The bar represents synchronisation of the completion of those activities with arcs into the transition.
- 8) **Decision and Merge Points:**
  - a. A decision point shows where the exit transition from a state or activity may branch in alternative directions depending on a condition.
  - b. A decision involves selecting one control-flow transition out of many control-flow transitions based on a condition
  - c. A merge point brings together alternate flows into a single output flow -

### How to construct Activity Diagrams

1. Finding system Actors, Classes and use cases
2. Identifying key scenarios of system use cases
3. Combining the scenarios to produce comprehensive work flows described using activity diagrams
4. Where significant object behaviour is triggered by a work flow, adding object flows to the diagrams
5. Where workflows cross technology boundaries, using swimlanes to map the activities

For the NextGEN POS system, the activity diagram is drawn as follows:



## 1.8 STATE MACHINE DIAGRAMS

State machine diagram specifies the sequence of states an object visits during its lifetime in response to 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.

### Purpose of state machine diagrams:

So the most important purpose of Statechart diagram is to model life time of an object from creation to termination.

Following are the main purposes of using Statechart diagrams:

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


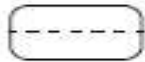



### State Machine view:

The **state machine** view describes the dynamic behavior of objects over time by modeling the lifecycles of objects of each class. Each object is treated as an isolated entity that communicates with the rest of the world by detecting **events** and responding to them.

### Elements of a state machine diagram:

1. **State:** A condition during the life of an object where it performs some activity.

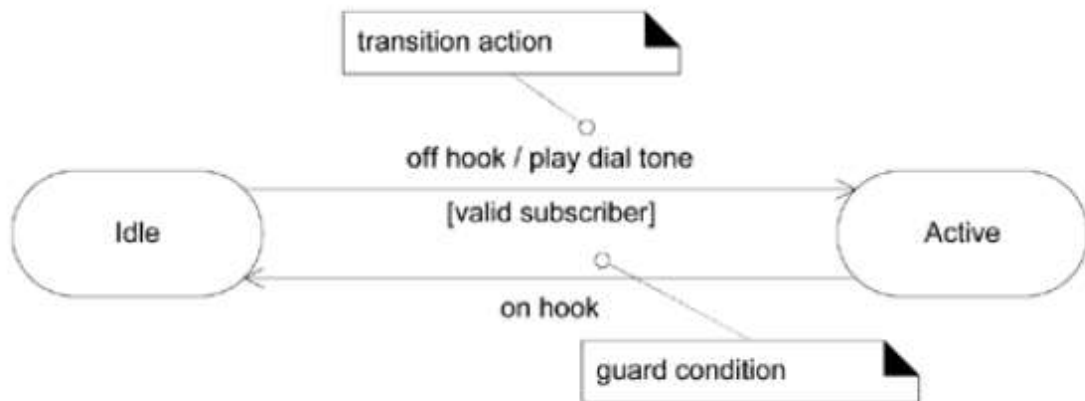
The different types of states are:

State Kind	Description	Notation
simple state	A state with no substructure	
concurrent composite state	A state that is divided into two or more concurrent substates, all of which are concurrently active when the composite state is active	
sequential composite state	A state that contains one or more disjoint substates, exactly one of which is active at one time when the composite state is active	
initial state	A pseudostate that indicates the starting state when the enclosing state is invoked	
final state	A special state whose activation indicates the enclosing state has completed activity	

2. **Event:** Specification of a significant occurrence. An event triggers a state transition.

3. **Transition:** A transition is a relationship between two states indicating that an object in the first state will, when a specified set of events and conditions are satisfied, perform certain actions and enter the second state.

**Figure 29.2. Transition action and guard notation.**



**Table 6-2: Kinds of Transitions and Implicit Actions**

Transition Kind	Description	Syntax
entry action	An action that is executed when a state is entered	entry/ action
exit action	An action that is executed when a state is exited	exit/ action
external transition	A response to an event that causes a change of state or a self-transition, together with a specified action. It may also cause the execution of exit and/or entry actions for states that are exited or entered.	e(a:T)[exp]/action
internal transition	A response to an event that causes the execution of an action but does not cause a change of state or execution of exit or entry actions	e(a:T)[exp]/action

4. **Entry and exit actions.** A transition across one or more levels of nesting may exit and enter states. A state may have actions that are performed whenever the state is entered or exited. Entering the **target state** executes an **entry action** attached to the state. If the **transition** leaves the original state, then its **exit action** is executed before the action on the transition and the entry action on the new state.
5. **Self-Transition:** A self-transition is a transition whose source and target states are the same.
6. **Completion transition.** A transition that lacks an explicit trigger event is triggered by the completion of activity in the state that it leaves (this is a **completion transition**).
7. **Substates:** A substate is a state that is nested in another state.
8. **Composite State:** A state that has substates is called a composite state.
9. **Simple State:** A state that has no substates is called a simple state.
10. **Action:** Task that takes place within a state.

Initial State

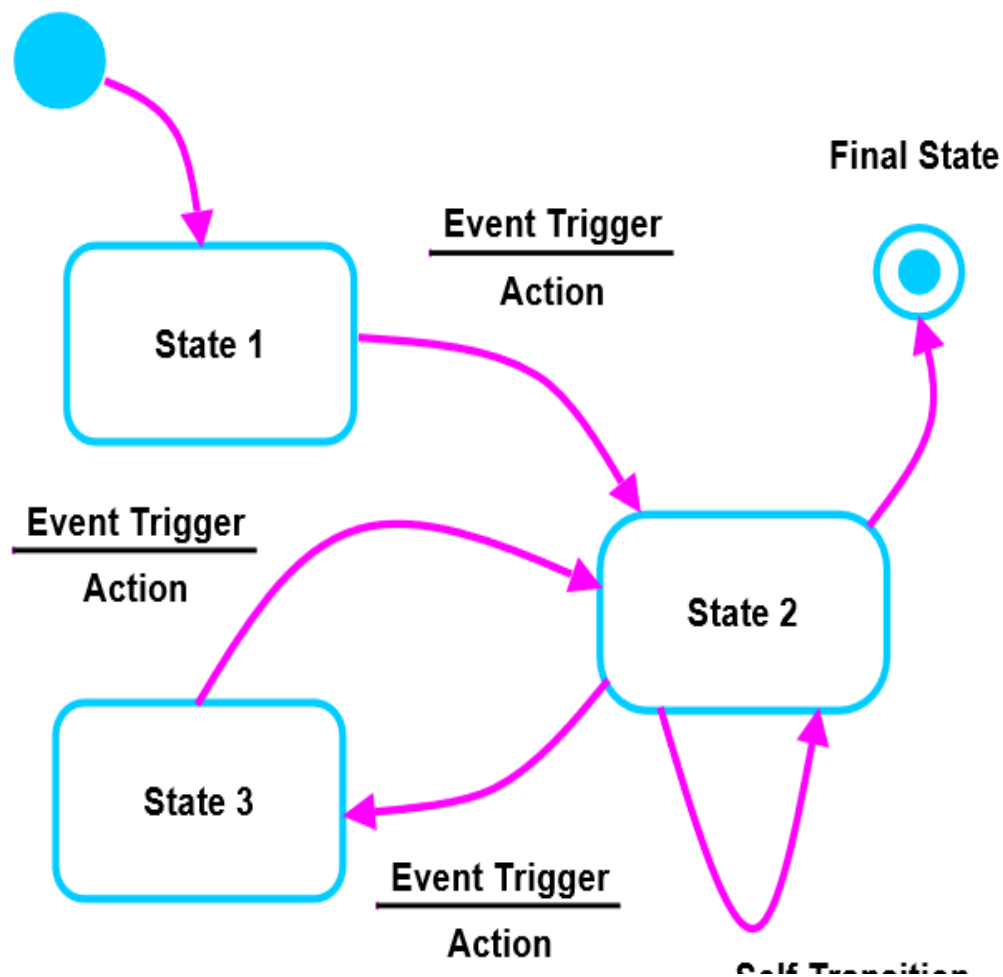


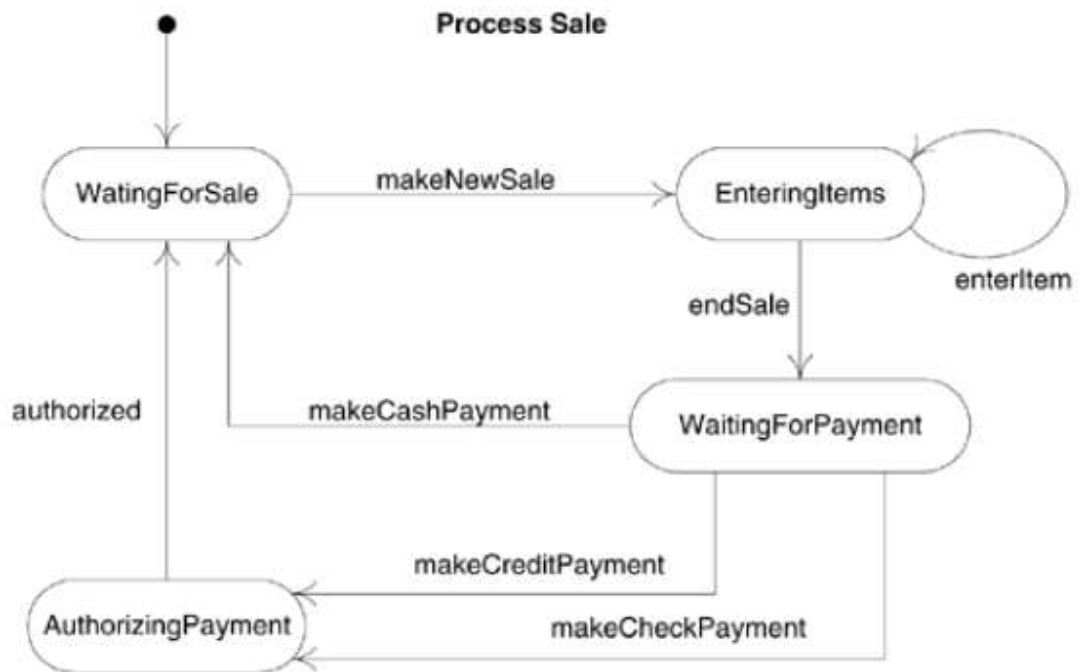
FIG : Basic Notations of a State Machine Diagram

## 29.3. How to Apply State Machine Diagrams?

### State-Independent and State-Dependent Objects

If an object always responds the same way to an event, then it is considered **state-independent** (or modeless) with respect to that event. For example, if an object receives a message, and the responding method always does the same thing. The object is state-independent with respect to that message. If, for all events of interest, an object always reacts the same way, it is a **state-independent object**. By contrast, **state-dependent objects** react differently to events depending on their state or mode.

For the NextGEN POS system, the state machine diagram is drawn as :



---

Library Management System

NextGen POS system

Airline reservation System

Telephone ringing [ state machine diagram]

---

Package

Component

Deployment diagrams from Mam's slides