

State Model

State model describes the sequences of operations for the objects that occur in response to external events.

The state model consists of multiple state diagrams.

The state diagram is a standard computer science concept that relates events and states.

State machine diagrams show how an object reacts to external events or conditions during the course of its lifetime

The State diagram is used to specify the behavior of objects in a class

A state diagram consists of

- States
- Events
- Transitions

Events represent external stimuli/occurrence and states represent values objects.

State:

- A state is a condition or situation during the life of an object in which it satisfies some condition, performs some activity, or waits for some event.
- A state is an abstraction of attribute values and links of a particular object
 - Sets of values and links are grouped together into a state according to the behaviour of objects
 - The objects in a class have a finite number of possible states.
 - Each object can be in one state at a time.
 - A state specifies the response of an object to input events
 - UML notation for state- a rounded box Containing an optional state name, list the state name in boldface, center the name near the top of the box, capitalize the first letter. e.g of states are as follows



Events:

- An event is an occurrence at a point in time
e.g. user depresses left button or Air Deccan flight departs from Bombay
- An event happens instantaneously with regard to time scale of an application.
- One event may logically precede or follow another, or the two events may be unrelated (concurrent; they have no effect on each other).
- Events include normal conditions as well as error conditions.

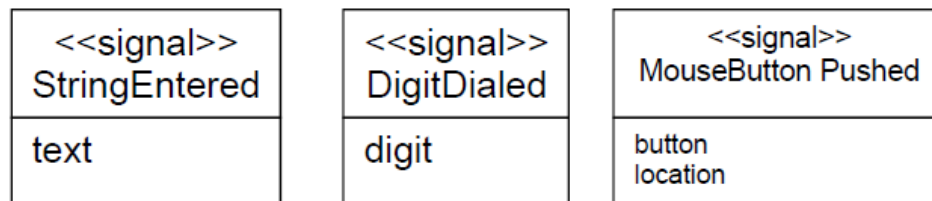
Three types of events:

- signal event,
- change event,
- time event.

Signal Event

- A signal is an explicit one-way transmission of information from one object to another.
- An object sending a signal to another object may expect a reply, but the reply is a separate signal under the control of the second object
- A signal event is the event of sending or receiving a signal (concern about receipt of a signal).

▪ Eg:



The difference between signal and signal event

- a signal is a message between objects
- a signal event is an occurrence in time.

Change Event

- A change event is an event that is caused by the satisfaction of a Boolean expression.
- UML notation for a change event is keyword **when** followed by a parenthesized Boolean expression.

Eg:

- when (room temperature < heating set point)
- when (room temperature > cooling set point)
- when (battery power < lower limit)
- when (tire pressure < minimum pressure)

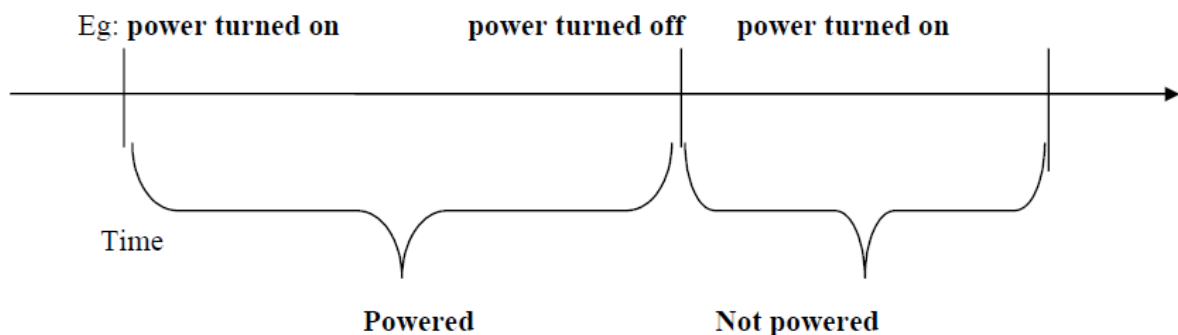
Time Event

- Time event is an event caused by the occurrence of an absolute time or the elapse of a time interval.
- UML notation for an absolute time is the keyword **when** followed by a *parenthesized expression involving time*.
- The notation for a time interval is the keyword **after** followed by a parenthesized expression that evaluates to a time duration.

- When (date=January 1,2022)
- After(10 seconds)

Event vs. States

- ☐ Event represents points in time.
- ☐ State represents intervals of time.

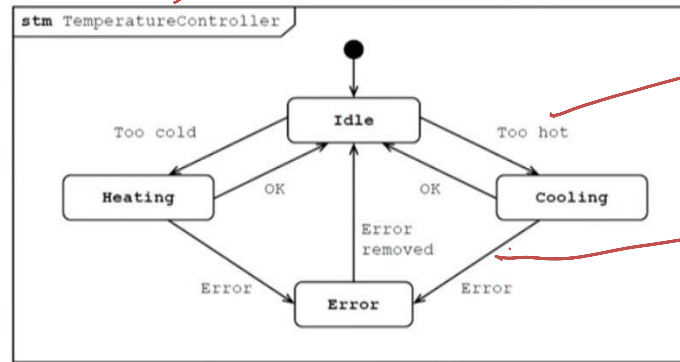


- A state corresponds to the interval between two events received by an object.
- The state of an object depends events.

Transitions & Guard Conditions

- A transition is the change from one state to another.
- A transition is represented by an arrow

stm stands for state machine

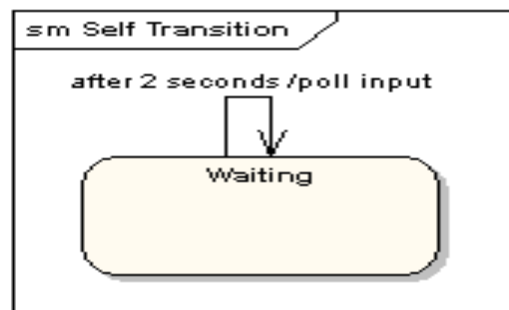


Event TooHot

Transition

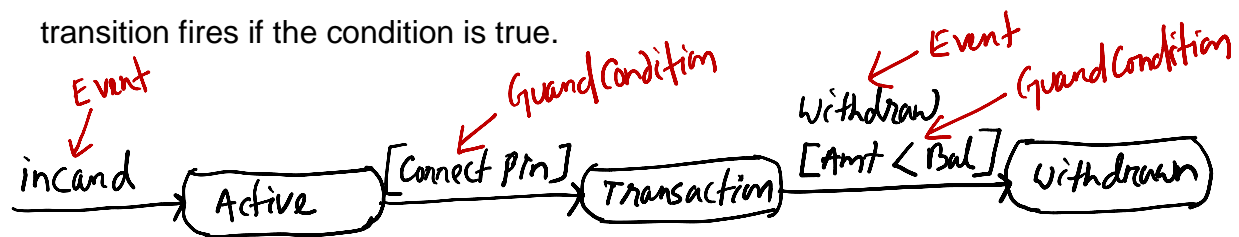
Self Transition:

- A self-transition is a transition that starts and ends in the same state. When a self-transition is taken, the state in question is exited and entered again.
- When an object's state doesn't change upon an event's occurrence, this is known as a self-transition i.e. in Self transition the source state and the destination state is same with response to the events



Guard Conditions

- In addition to the event happening, the guard condition must also be true for the transition to take place”
- A guard condition is a Boolean expression that must be true in order for a transition to occur.
- A guard condition is checked only once, at the time the event occurs, and the transition fires if the condition is true.

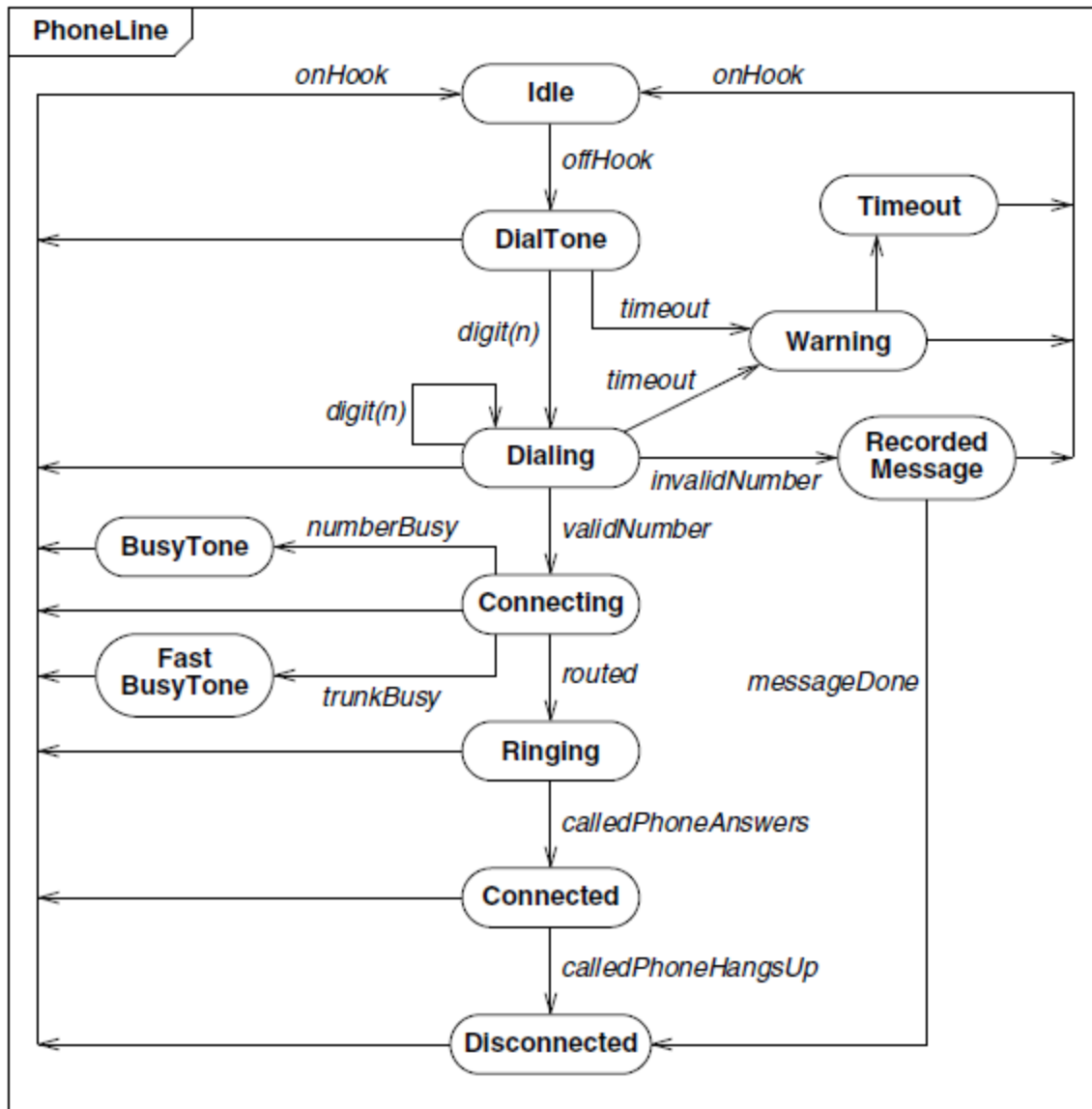


The above figure shows Guarded transitions.

State Diagram

- A state diagram is a graph whose nodes are states and whose directed arcs are transitions between states.
- A state diagram specifies the state sequence caused by event sequences.
- All objects in a class execute the state diagram for that class, which models their common behavior.
- A state model consists of multiple state diagrams, one state diagram for each class with important temporal behavior.
- UML notation for a state diagram is a rectangle with its name in small pentagonal tag in the upper left corner.

The following example shows state diagram for a TelephoneLine



Actions and Activities within a State

Activities are ongoing operations; actions are instantaneous operations

Different actions and activities within a state are as follows

- **Entry action/activity** - Upon each entry to a state, a specified action is automatically executed.

Syntax : **entry / action**

- **Exit action/activity** - Just prior to leaving a state, a specified action is automatically executed.

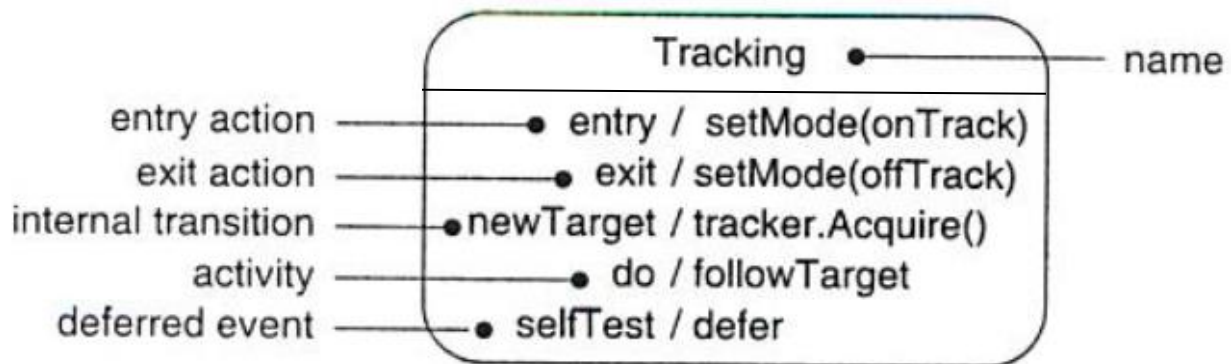
Syntax : **exit / action**

- **Internal Transitions** - The handling of an event without leaving the current state.
Used to avoid a state entry and exit actions.

Syntax : **event / action**

- **do activities** - Ongoing work that an object performs while in a particular state.
The work automatically terminates when the state is exited.

Syntax : **do / activity**

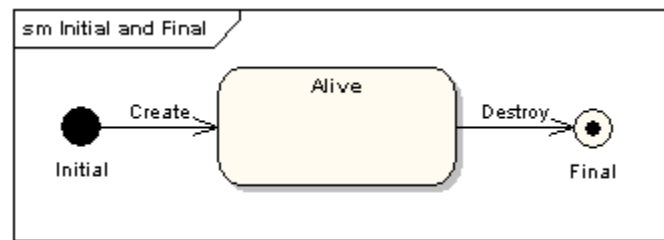


- **Deferred Event** - An event whose occurrence is responded to at a later time.

Syntax : **event / defer**

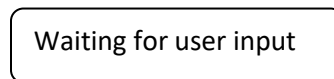
State Diagram concepts and notations:

- **Initial state** - indicates the initial starting state for the state machine or a substate.
- **Final state** - indicates the state machine's execution has completed.



Simple State:

A state that does not have any substates.



e.g. Simple state without representing activities



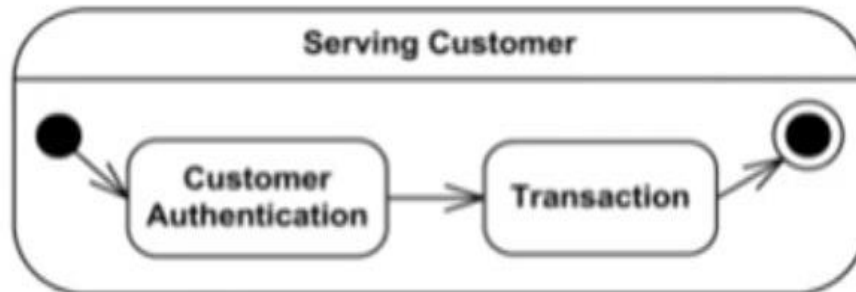
e.g. Simple state with entry and exit activities

Composite state - A state that contains *substates*.

Substate :

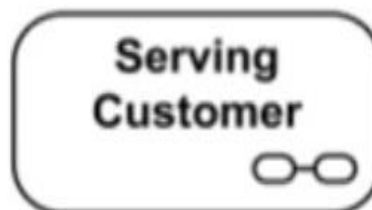
- A state that is nested inside another state.
- Substates allow state diagrams to show different levels of abstraction.
- Substates may be sequential or concurrent.
- Substates may be nested to any level.

- A **composite** state is defined as state that has substates (nested states)
- Substates could be **sequential (disjoint)** or **concurrent (orthogonal)**
- A composite state can have one or more **regions**
- A **region** contains states and transitions
- **Simple composite state** contains just one region



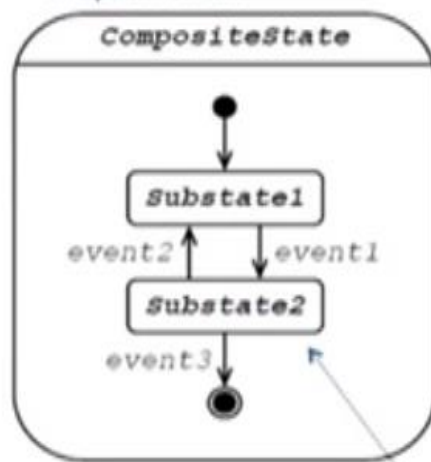
Simple composite state Serving Customer has two substates

- **Orthogonal composite state** has more than one regions
- Any state enclosed within a region of a composite state is called a substate of that composite state
- A composite state has an additional **decomposition compartment** apart from the initial 3 compartments
- **Decomposition compartment** shows composition structure of the state consisting of regions, states, and transitions

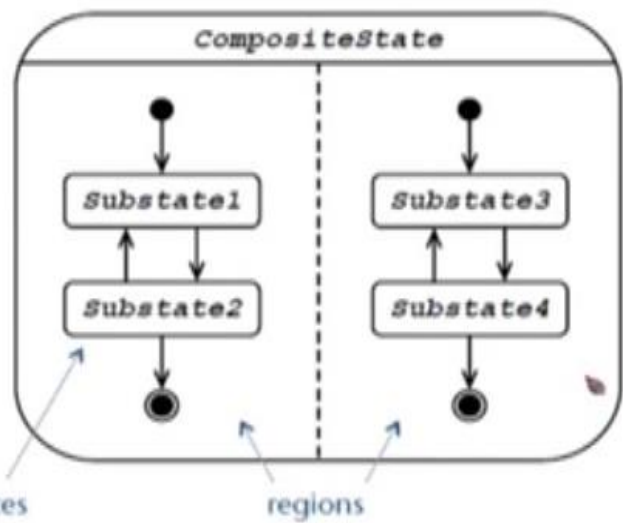


Composite state Serving Customer with decomposition hidden

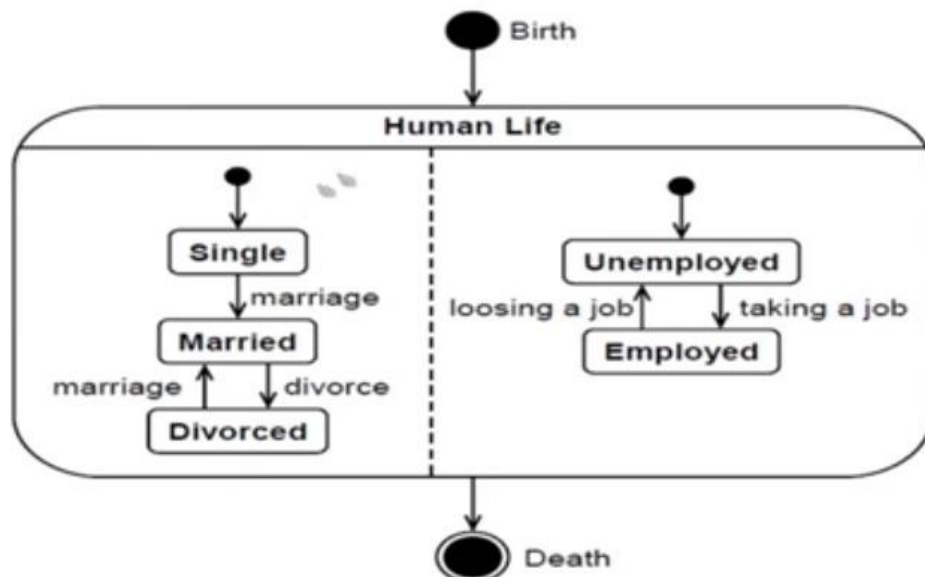
Non-orthogonal
composite state:



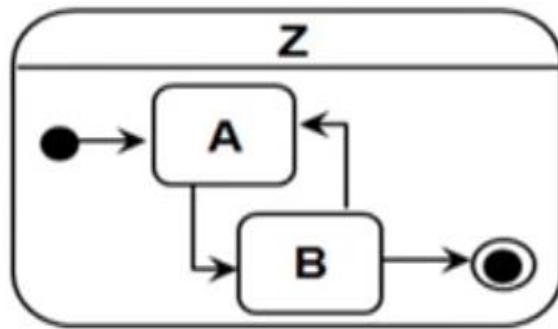
Orthogonal composite state:



e.g. Composite State

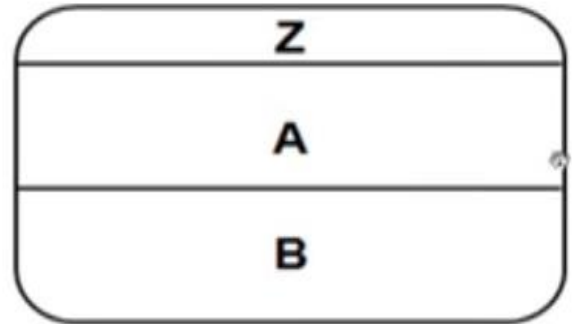


disjoint sub-states (OR-Refinement): Exactly one substate is active when the superstate is active



Either A or B is active, when Z is active

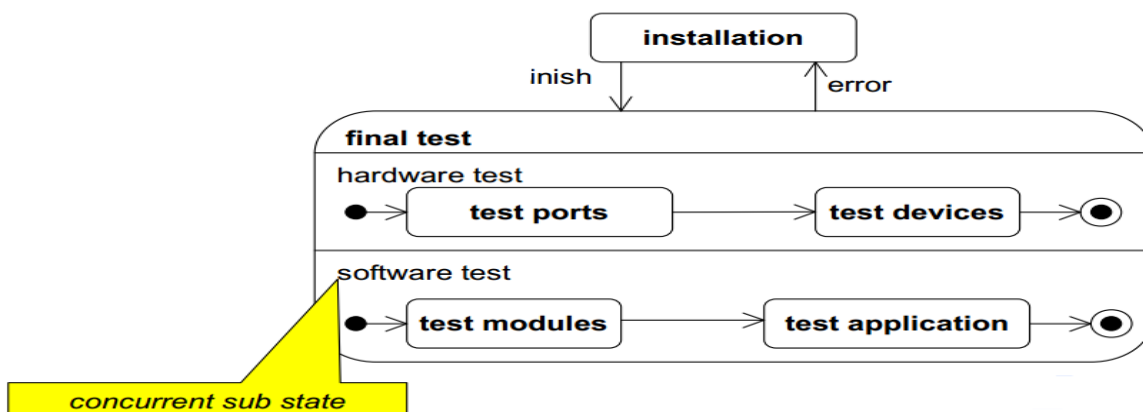
parallel sub-states (AND-Refinement): All substates are active when the superstate is active



Both A and B are active, when Z is active

Concurrent Regions :

- A state may be divided into regions containing substates that exist and execute concurrently.
- The example below shows concurrent regions executing simultaneously.



Pseudo State :

- Pseudostates (abstract vertex) are typically used to connect multiple transitions into more complex state transitions paths
- Pseudostates include
 - initial pseudostate
 - terminate pseudostate
 - entry point
 - exit point
 - choice
 - join
 - fork
 - junction
- History state

Initial pseudostate: Source for a single transition to the default state of a composite state.

Notation: Small solid filled circle



Initial pseudostate transitions to Waiting for User
Input state

Terminal pseudostate: implies termination of execution of the state.

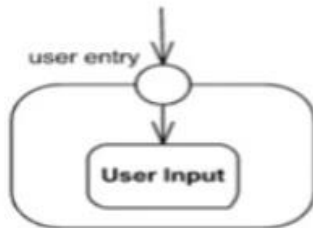
Notation: cross



Transition to terminate pseudostate

Entry point pseudostate: is an entry point of a state machine or composite state.

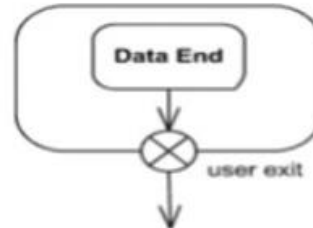
Notation: small circle on the border of the state machine diagram



Entry point user entry

Exit point pseudostate: is an exit point of a state machine or composite state.

Notation: small circle with a cross on the border of the state machine diagram

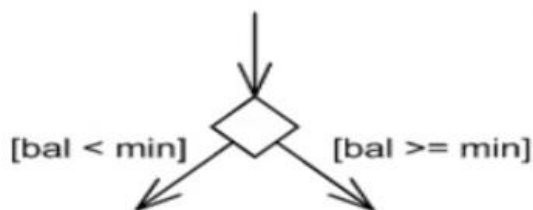


Exit point user exit

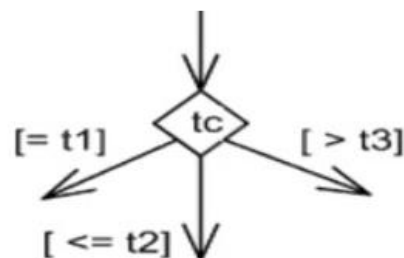
Choice Pseudo-State:

A choice pseudo-state is shown as a diamond with one transition arriving and two or more transitions leaving. The following diagram shows that whichever state is arrived at, after the choice pseudostate, is dependent on the message format selected during execution of the previous state.

Notation: Diamond shaped symbol

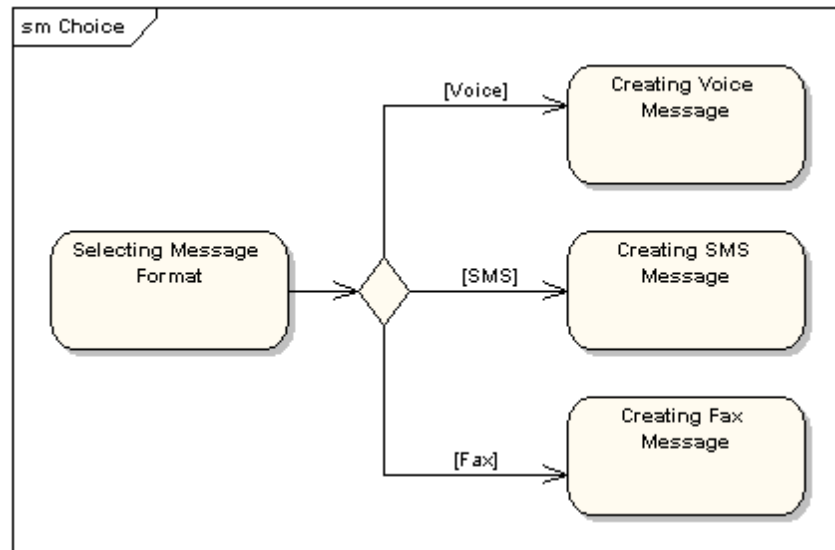


Select outgoing transition based on condition



Choice based on guards applied to the value inside diamond

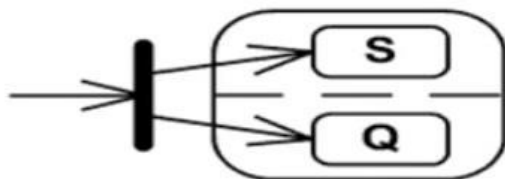
e.g. pseudostates



Fork and Join Pseudostate :

Fork pseudostate : splits an incoming transition into two or more transitions terminating on target vertices in different regions .

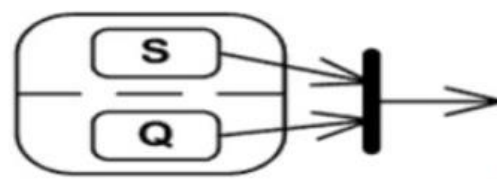
Notation: short heavy bar



Fork splits transition into two transitions

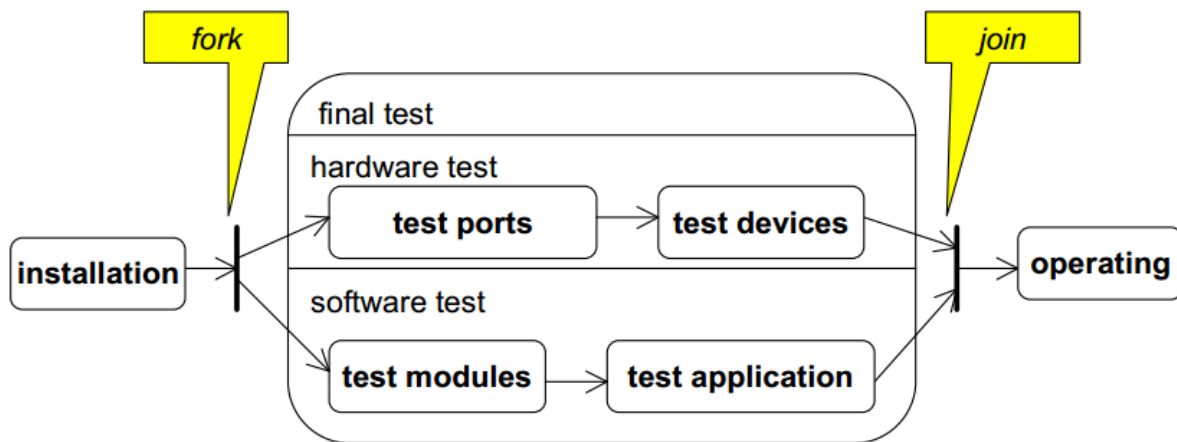
Join pseudostate : merges several transitions originating from source vertices in different regions .

Notation: short heavy bar



Join merges transitions into single transition

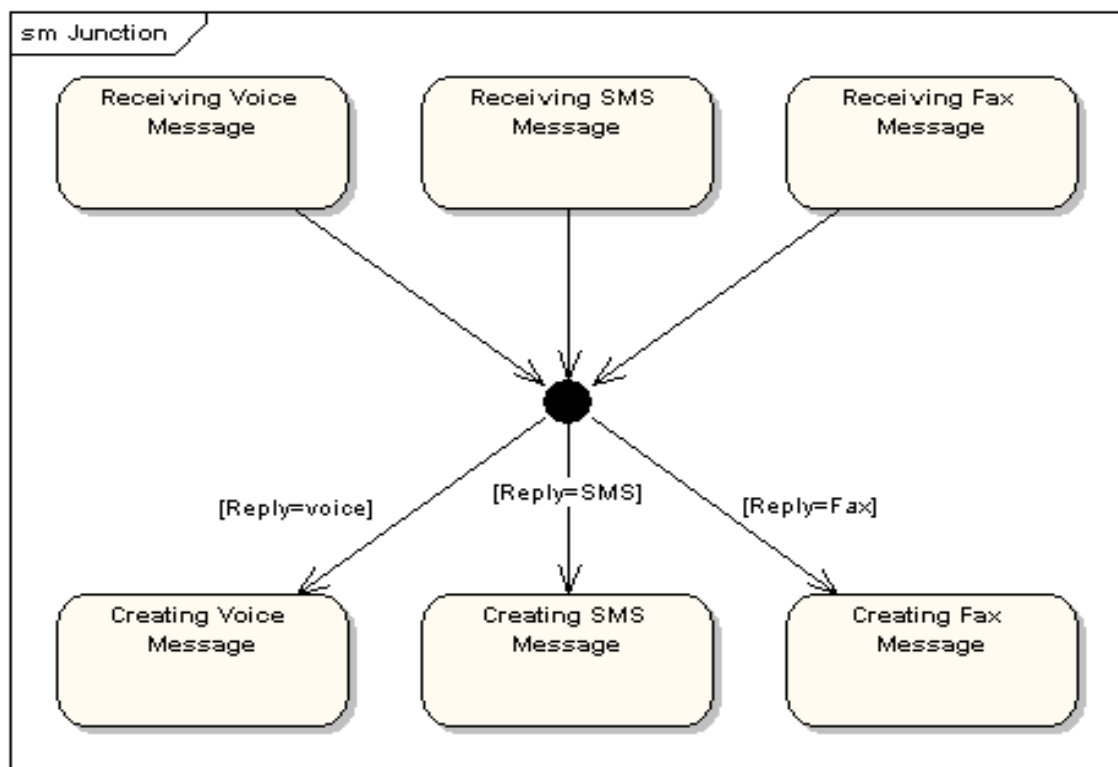
e.g. Fork and Join Pseudostate



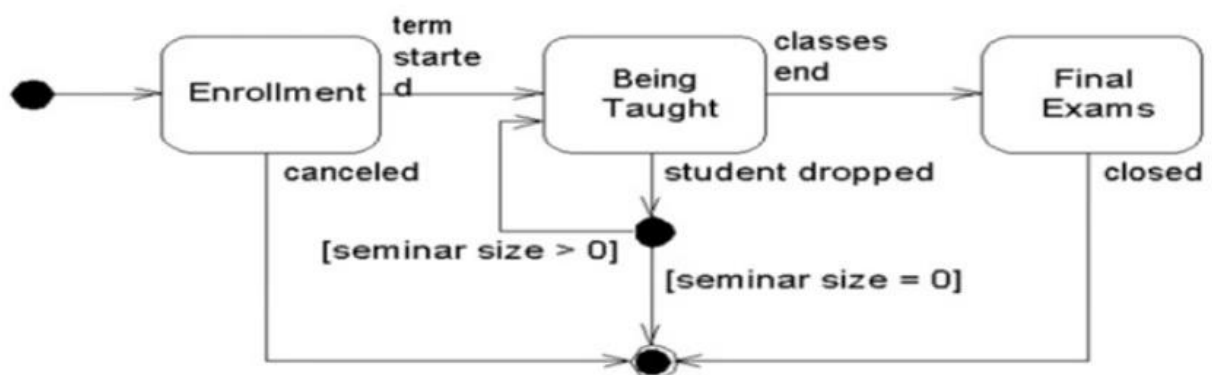
Junction Pseudo-State :

- Junction pseudo-states are used to chain together multiple transitions.
- A single junction can have one or more incoming, and one or more outgoing, transitions; a guard can be applied to each transition.
- Junctions are semantic-free.
- A junction which splits an incoming transition into multiple outgoing transitions realizes a static conditional branch, as opposed to a choice pseudo-state which realizes a dynamic conditional branch.


e.g. junction pseudostate

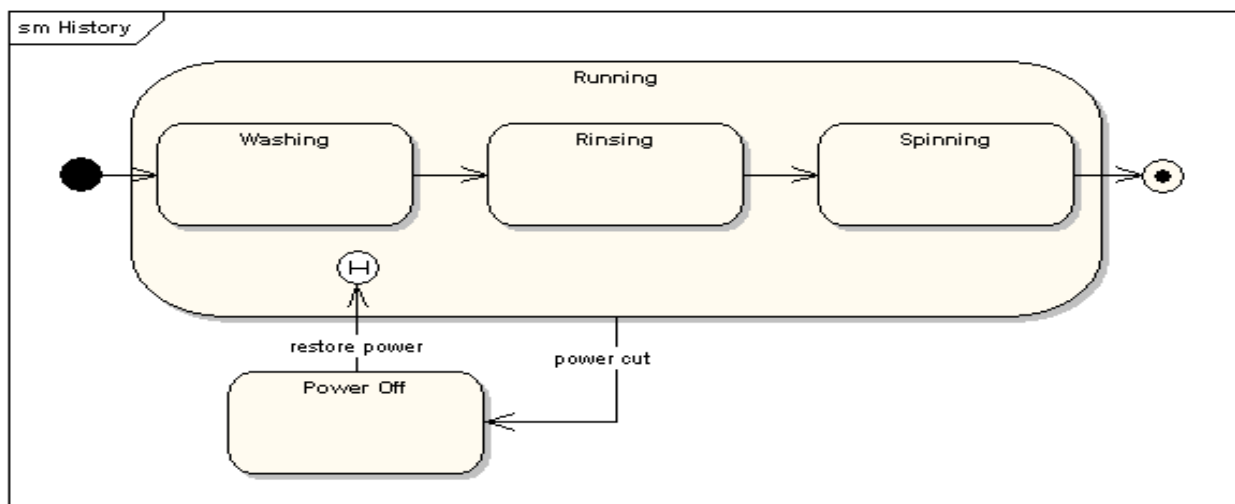


e.g. Junction pseudostate



History States

- A history state is used to remember the previous state of a state machine when it was interrupted.
- Notation 
- The following diagram illustrates the use of history states.
- The example is a state machine belonging to a washing machine.



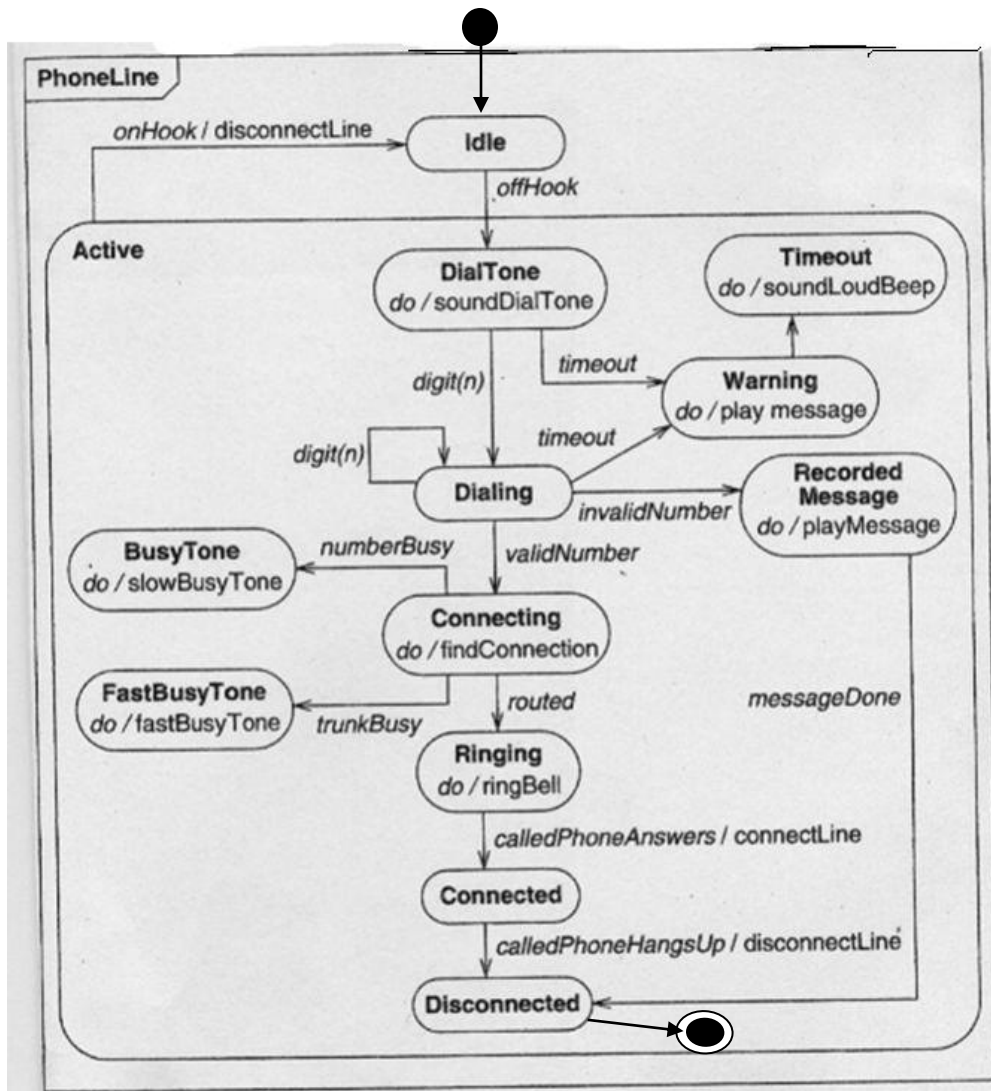
Final State

- Final state indicates that the state machine's execution has completed.
- **Final state** is a special kind of state signifying that the enclosing region is completed.
- *Notation:* circle surrounding a small solid filled circle

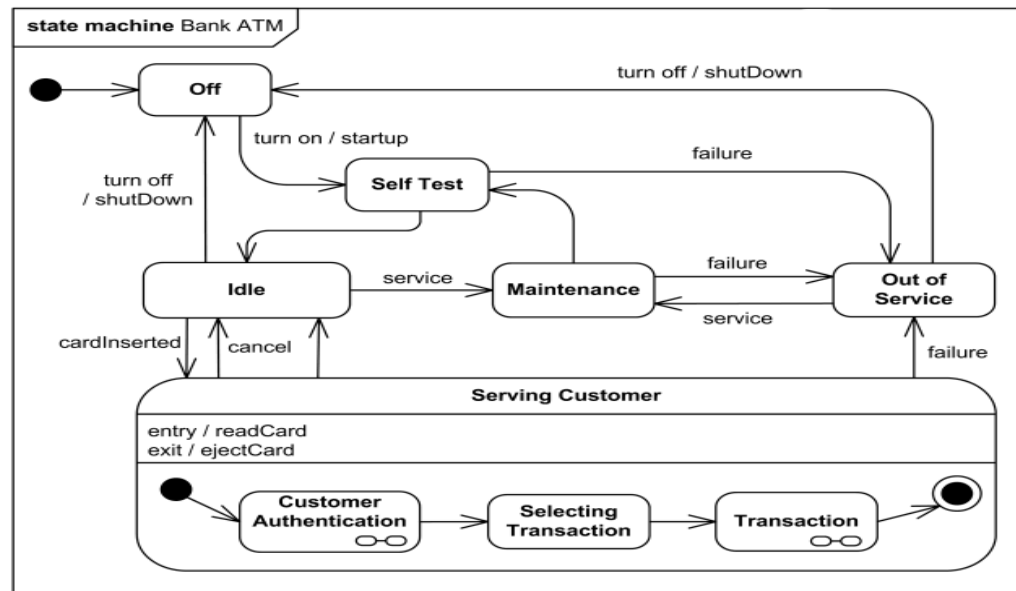


Transition to final state

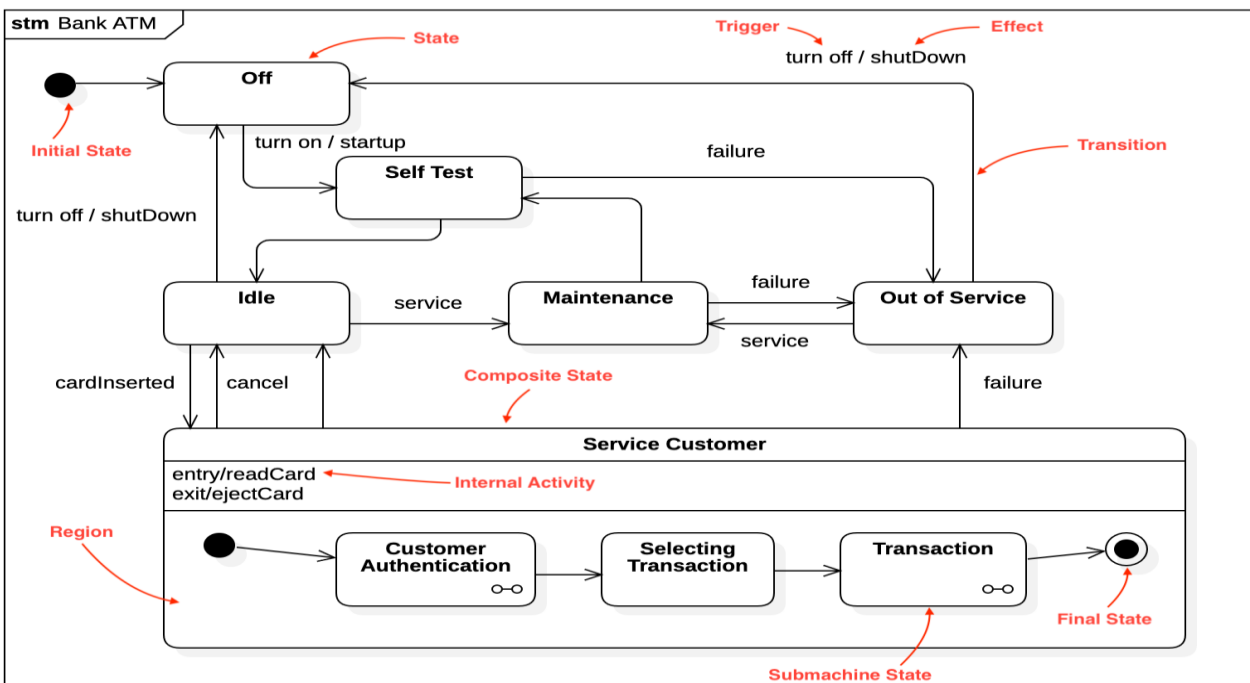
e.g State Diagram of a Telephone line



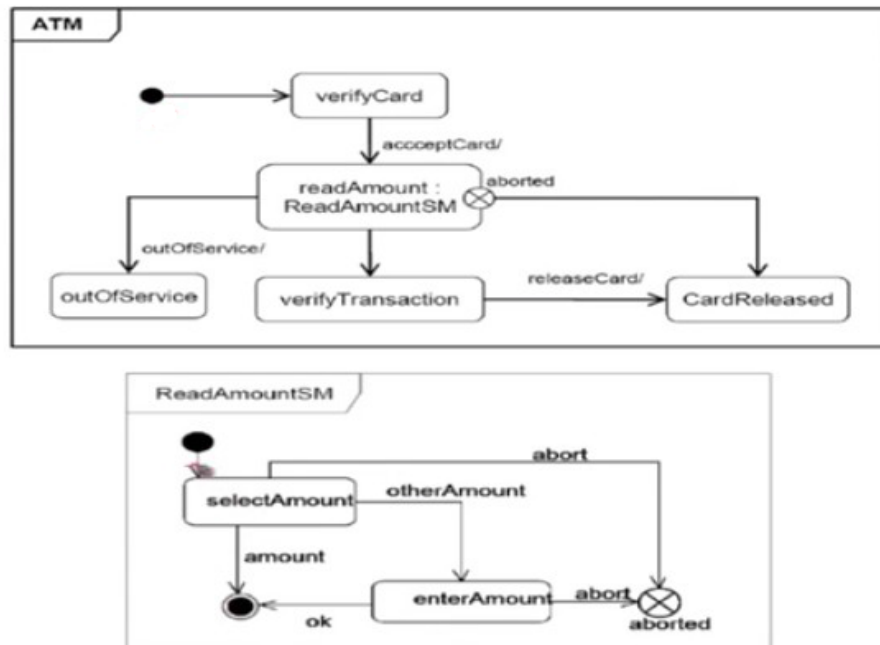
e.g. State Diagram for an ATM Machine



Or e.g. State Diagram for an ATM Machine explanation



e.g State Diagram for an ATM Machine (in another way)



e.g. State diagram of online order processing with explanation

