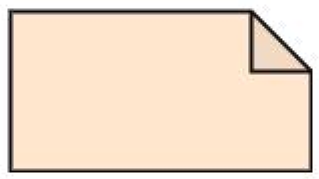
Note

A note is the capability of attaching several remarks to the element. It basically carries useful information for the modelers.



Events--

An event is an occurrence at a point in time, such as *user depresses left button* or *flight 123 departs from Chicago ,power turned on, alarm set, paper tray becomes empty, temperature becomes lower than freezing.*

Two events that are causally unrelated are said to be *concurrent:* they have no effect on each other. Flight 123 must depart Chicago before it can arrive in San Francisco; the two events are causally related. Flight 123 may depart before or after flight 456 departs Rome; the two events are causally unrelated.

There are several kinds of events. The most common are the signal event, the change event, and the 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, which may or may not choose to send it.

We can give each signal class a name to indicate common structure and behavior. For example,

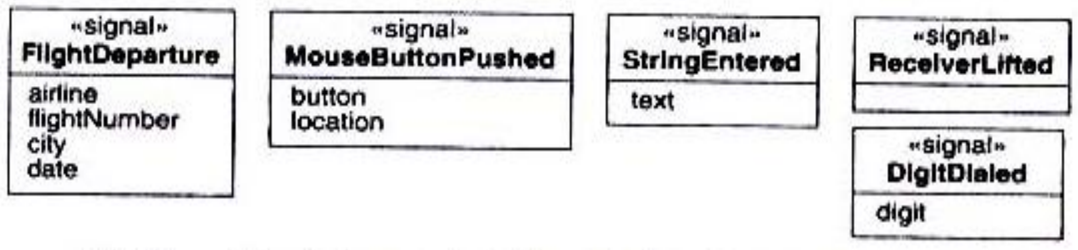
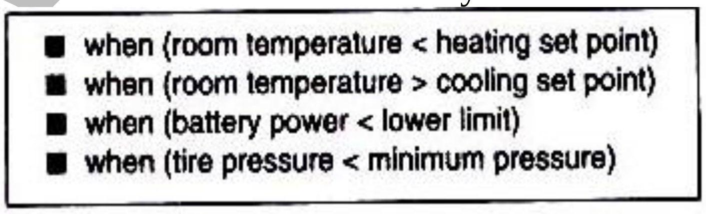


Figure **Signal classes and attributes.** A signal is an explicit one-way transmission of information from one object to another.

Change Event--

A *change event* is an event that is caused by the satisfaction of a boolean expression.

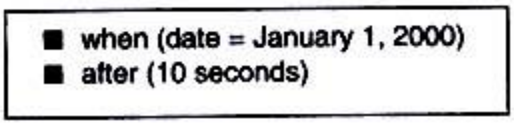


- when (room temperature < heating set point)
- when (room temperature > cooling set point)
- when (battery power < lower limit)
- when (tire pressure < minimum pressure)

Figure **Change events.** A change event is an event that is caused by the satisfaction of a boolean expression.

Time Event--

A *time event* is an event caused by the occurrence of an absolute time or the elapse of a time interval.



- when (date = January 1, 2000)
- after (10 seconds)

Figure **Time events.** A time event is an event caused by the occurrence of an absolute time or the elapse of a time interval.

States--

A *state* is an abstraction of the values and links of an object. Sets of values and links are grouped together into a state according to the gross behavior of objects. Figure 5.4 shows the UML notation for a state—a rounded box containing an optional state name.

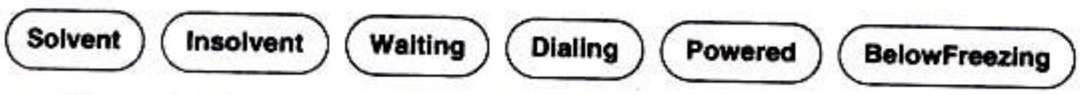


Figure **States.** A state is an abstraction of the values and links of an object.

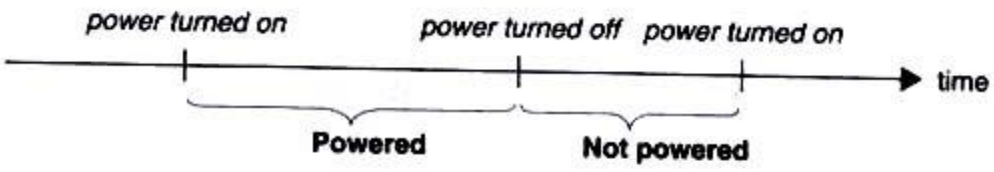
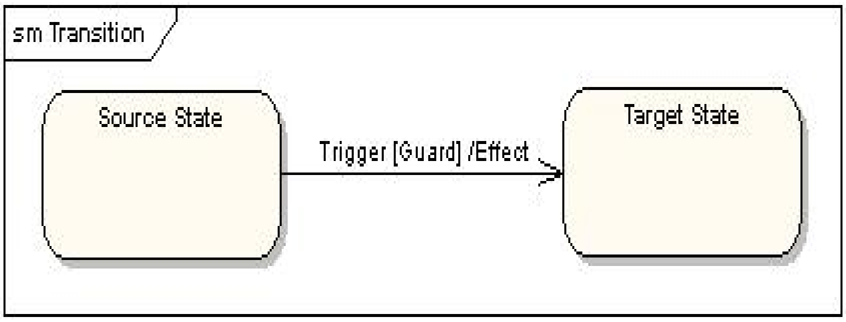


Figure **Event vs. state.** Events represent points in time; states represent intervals of time.

Guard--

Transitions from one state to the next are denoted by lines with arrowheads. A transition may have a trigger, a guard and an effect, as below.



"Trigger" is the cause of the transition, which could be a signal, an event, a change in some condition, or the passage of time. "Guard" is a condition which must be true in order for the trigger to cause the transition. "Effect" is an action which will be invoked directly on the object that owns the state machine as a result of the transition.

A *guard condition* is a boolean expression that must be true in order for a transition to occur.

State Diagrams--

A *state diagram* is a graph whose nodes are states and whose directed arcs are transitions between states. A state diagram specifies the state sequences caused by event sequences.

Sample State Diagram--

Figure shows a state diagram for a telephone line. The diagram concerns a phone line and not the caller nor callee. The diagram contains sequences associated with normal calls as well as some abnormal sequences, such as timing out while dialing or getting busy lines.

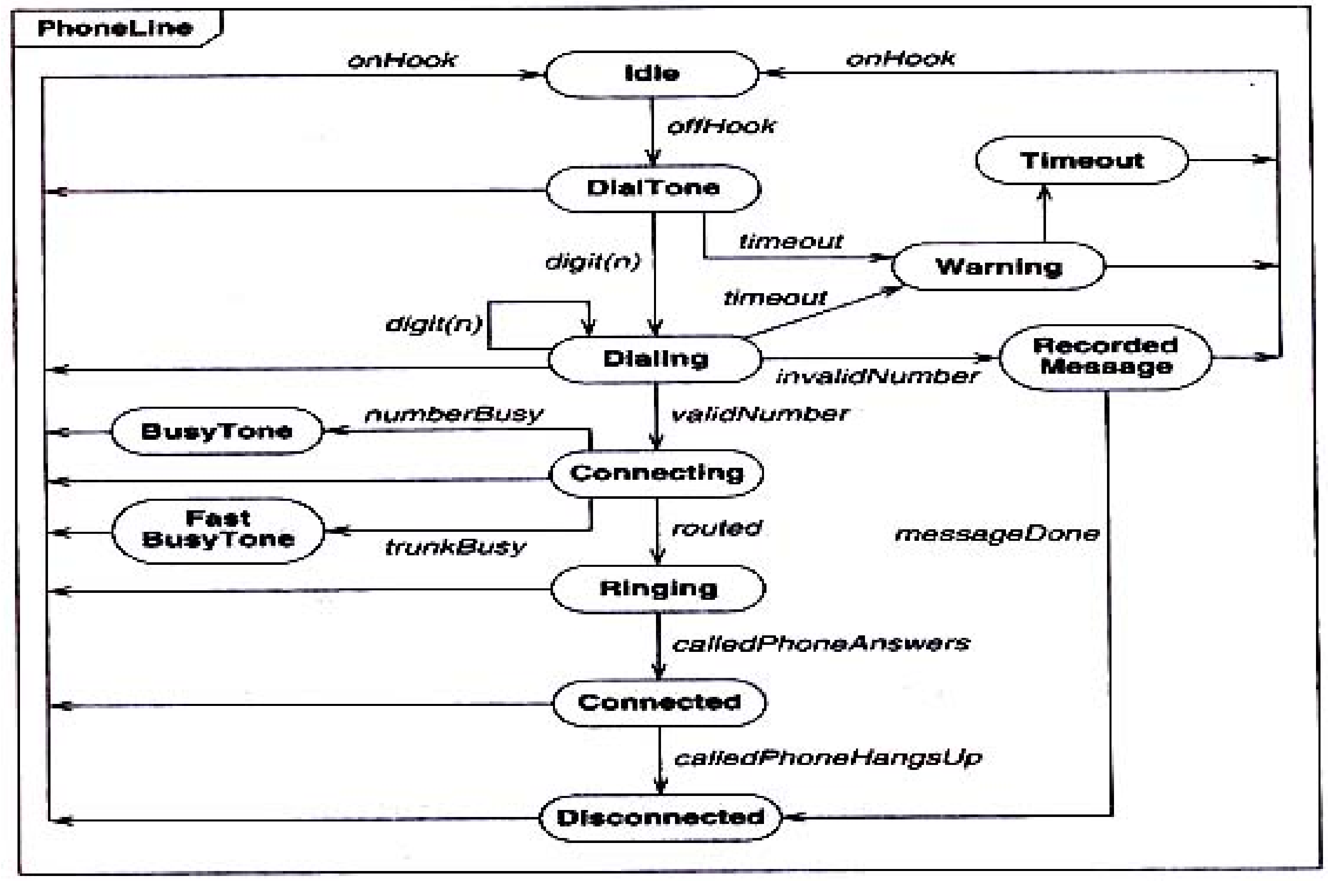


Figure **State diagram for a telephone line.** A state diagram specifies the state sequences caused by event sequences.

One-shot State Diagrams--

One-shot state diagrams represent objects with finite lives and have initial and final states. The initial state is entered on creation of an object; entry of the final state implies destruction of the object. Figure 5.9 shows a simplified life cycle of a chess game with a default initial state (solid circle) and a default final state (bull's eye).

As an alternate notation,We can indicate initial and final states via entry and exit points. In Figure the *start* entry point leads to white's first turn, and the chess game eventually ends with one of three possible outcomes. Entry points (hollow circles) and exit points (circles enclosing an "x") appear on the state diagram's perimeter and may be named.

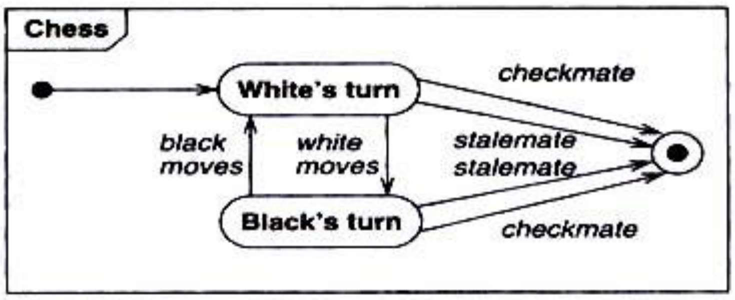


Figure **State diagram for chess game.** One-shot diagrams represent objects with finite lives.

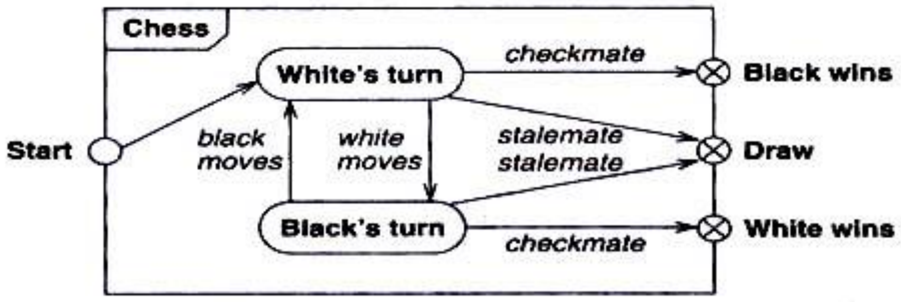


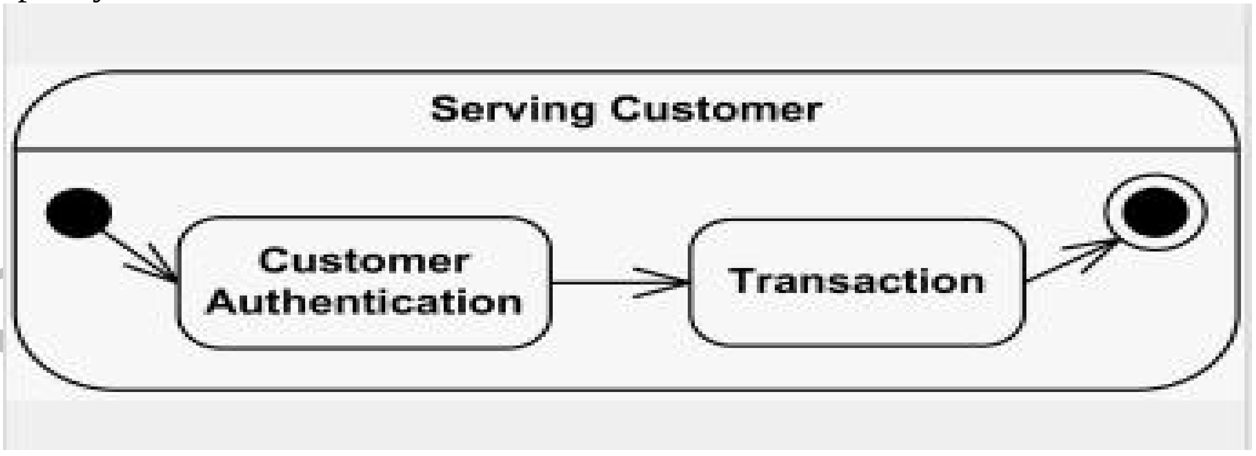
Figure **State diagram for chess game.** You can also show one-shot diagrams by using entry and exit points.

<u>Composite state--</u>

A composite state is a state that contains one or more other substates. It can be used to group states into logical compounds and thus make the statechart more comprehensible.

When a composite state is entered, its entry node denotes the substate to be activated. A composite state can specify multiple entry nodes with unique names. Incoming transitions of the composite state can specify the desired entry node to take for entering the composite state.

When a composite state is left, all active substates are also left. A composite state can specify multiple exit nodes with unique names. Outgoing transitions of the composite state can specify the relevant exit node for them.



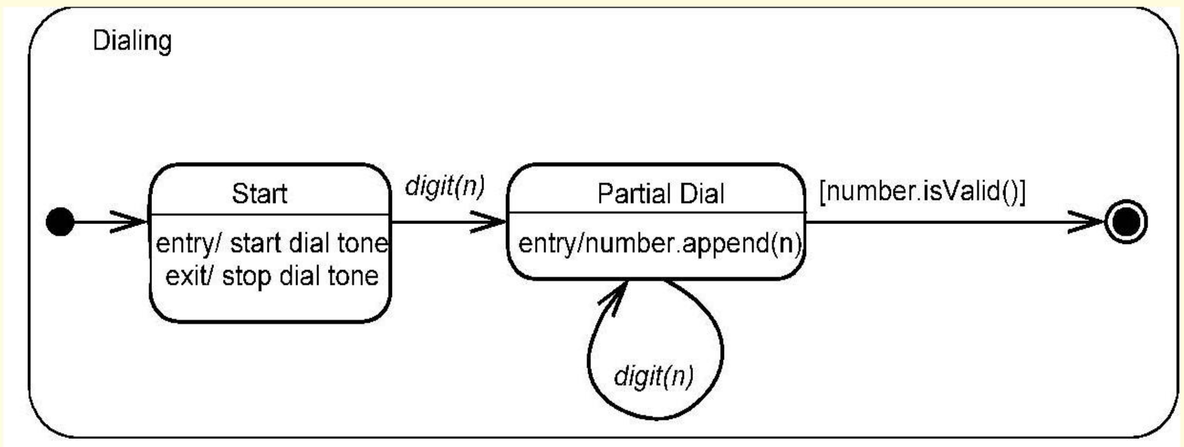


Figure - Composite state with two states

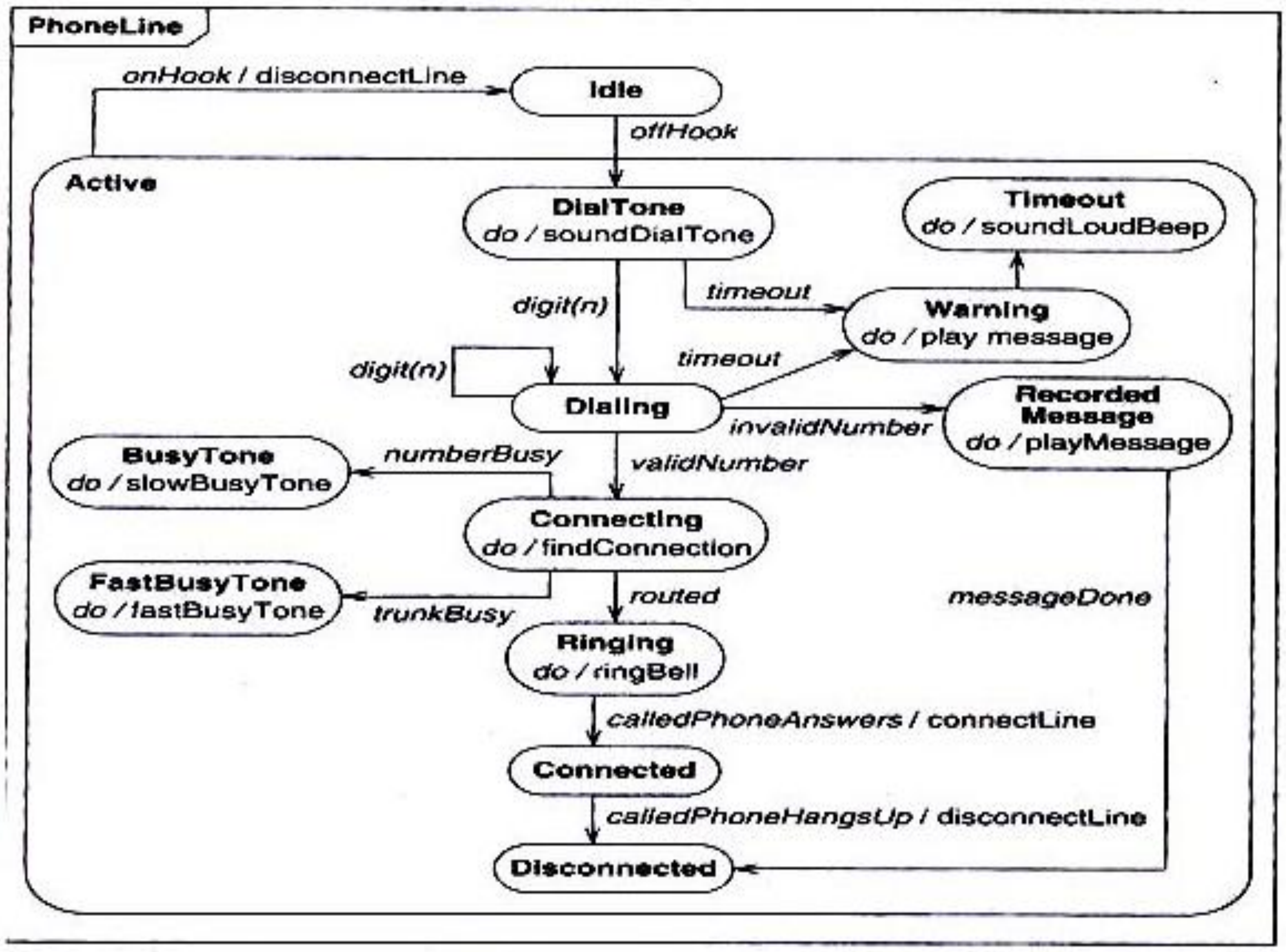
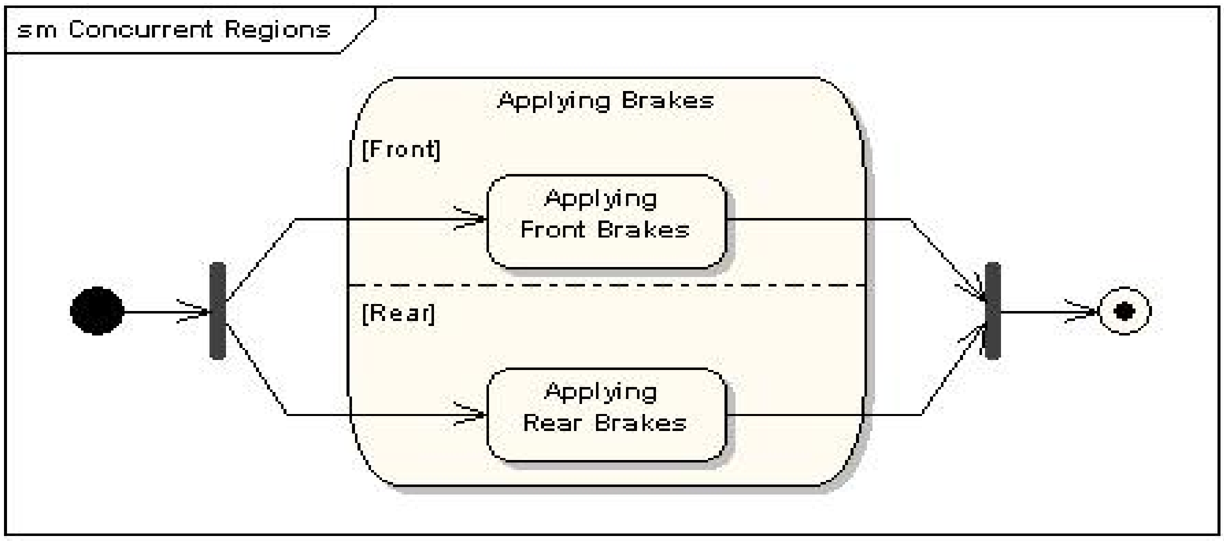


Figure **Nested states for a phone line.** A nested state receives the outgoing transitions of its enclosing state.

Concurrent States--

A state may be divided into regions containing sub-states that exist and execute concurrently. The example below shows that within the state "Applying Brakes", the front and rear brakes will be operating simultaneously and independently. Notice the use of fork and join pseudo-states, rather than choice and merge pseudo-states. These symbols are used to synchronize the concurrent threads.



Examples—

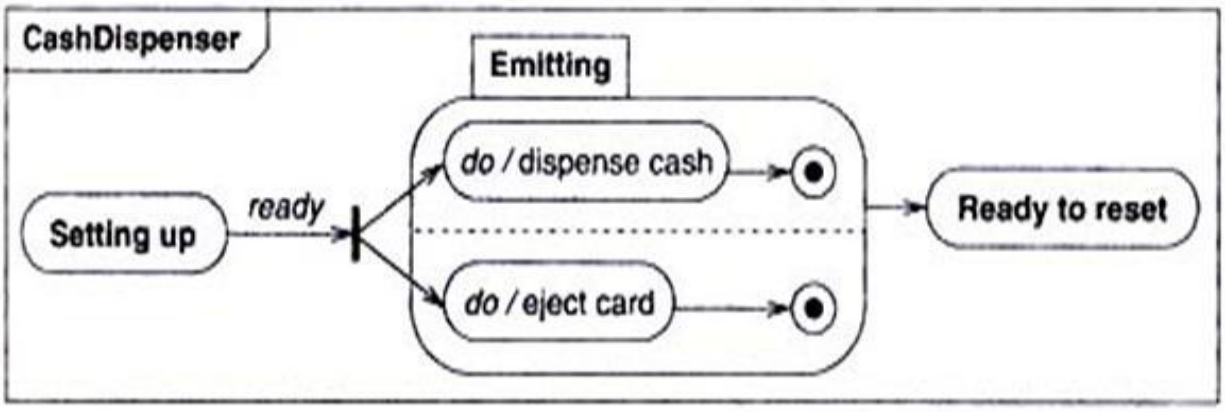
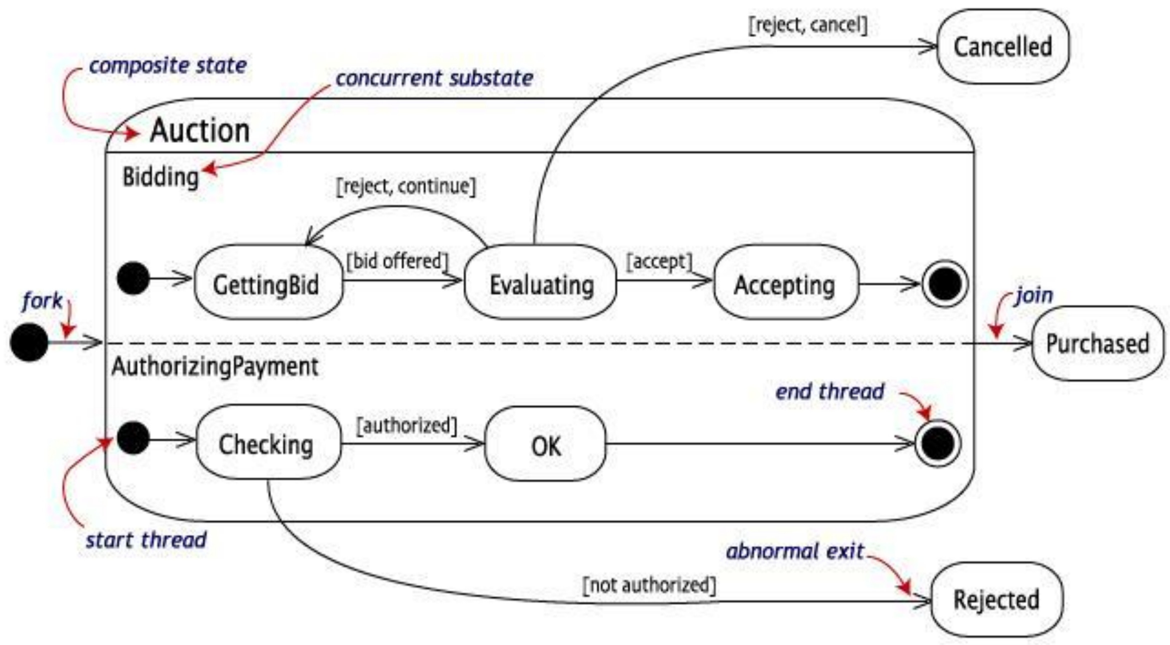
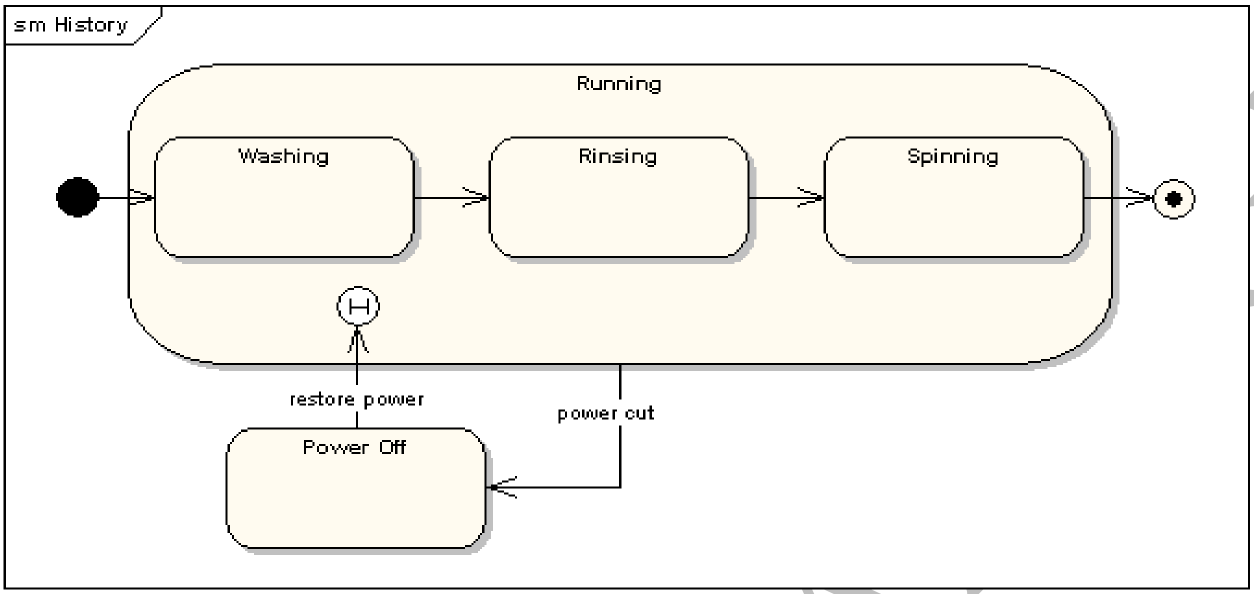


Figure **Synchronization of control.** Control can split into concurrent activities that subsequently merge.



History States--

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



In this state machine, when a washing machine is running, it will progress from "Washing" through "Rinsing" to "Spinning". If there is a power cut, the washing machine will stop running and will go to the "Power Off" state. Then when the power is restored, the Running state is entered at the "History State" symbol meaning that it should resume where it last left-off.

Example—

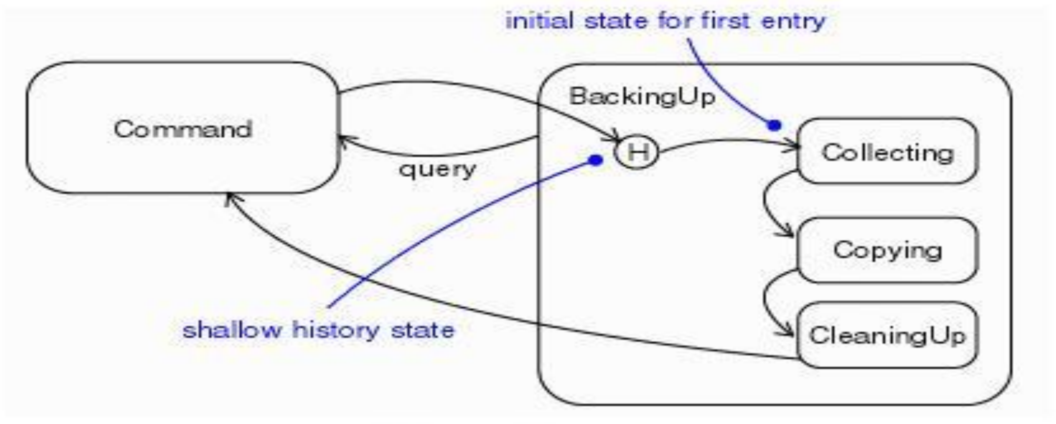
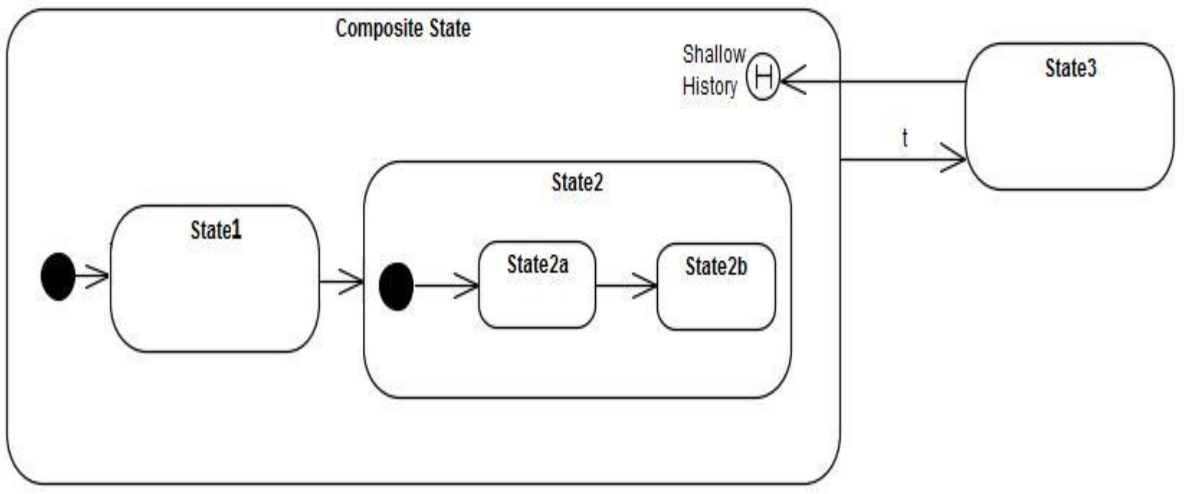


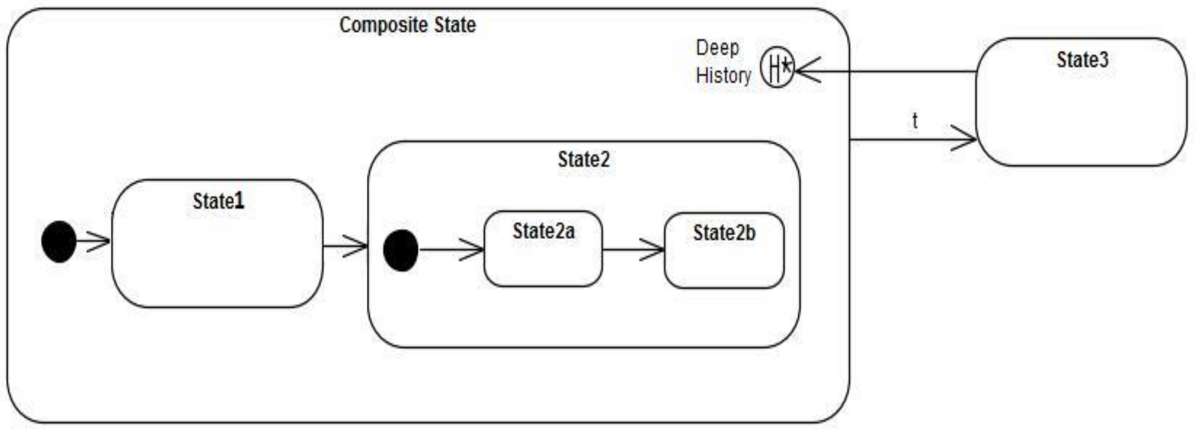
Figure . History State.

Shallow History--



- Represents the most recent **active substate** of its containing Region, but not the substates of that substate.
- Transition terminating on this Pseudostate implies restoring the Region to that substate.
- A single outgoing Transition from this Pseudostate may be defined terminating on a substate of the composite State.
- Shallow History can only be defined for composite States and, at most one such Pseudostate can be included in a Region of a composite State.

Deep History--



- Represents the most recent **active state configuration** of its owning Region.
- Transition terminating on this Pseudostate implies restoring the Region to that same state configuration.
- The entry Behaviors of all States in the restored state configuration are performed in the appropriate order starting with the outermost State.
- Deep History Pseudostate can only be defined for composite States and, at most one such Pseudostate can be contained in a Region of a composite State.