State Modeling

- State model describes the sequences of operations that occur in response to external stimuli.
- The state model consists of multiple state diagrams, one for each class with temporal behavior that is important to an application.
- The state diagram is a standard computer science concept that relates events and states.
- Events represent external stimuli and states represent values objects.

Events:

- An event is an occurrence at a point in time, such as 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 error conditions as well as normal conditions.

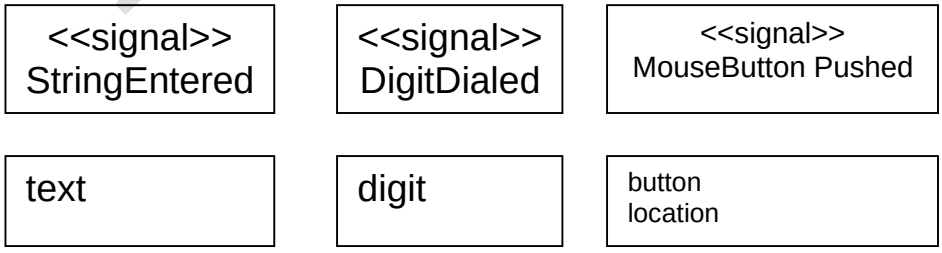
Three types of events:

- Signal event
- Change event
- Time event.

1. Signal Event

- A signal is an explicit one-way transmission of information from one object to another.
- It is different form a subroutine call that returns a value.
- 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.
- 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.

2. 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)

3. 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.

Eg:

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

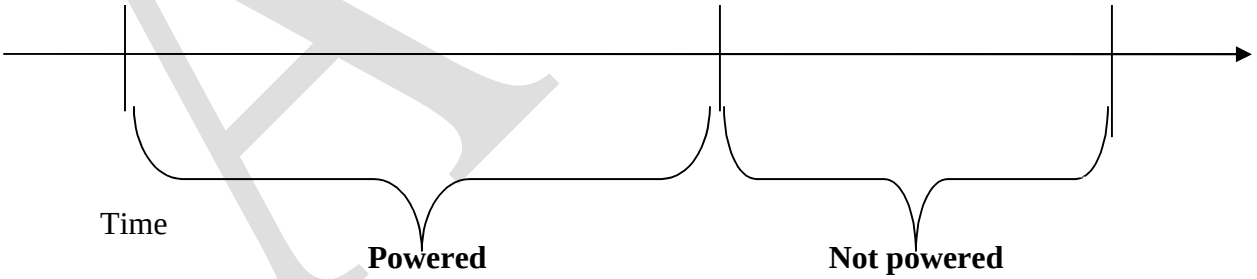
States:

- 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
- 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.
- Ignore attributes that do not affect the behavior of the object.
- 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.
- All events are ignored in a state, except those for which behavior is explicitly prescribed.

Event vs. States

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

Eg: **power turned on** **power turned off** **power turned on**



- A state corresponds to the interval between two events received by an object.
- The state of an object depends on past events.
- Both events and states depend on the level of abstraction.
- A state specifies the response of an object to input events

State Alarm ringing on a watch

- **State** : *Alarm Ringing*
- **Description** : alarm on watch is ringing to indicate target time
- **Event sequence that produces the state** :
  setAlarm (targetTime)
  any sequence not including *clearAlarm*
  when (*currentTime = targetTime*)
- **Condition that characterrizes the state:**
  alarm = on, alarm set to *targetTime*,
  targetTime <= currentTime <=targetTime+20 sec , and no button has been pushed since *targetTime*
- **Events accepted in the state:**

event	response	next state
when (*currentTime = targetTime+20*)	*resetAlarm*	*normal*
buttonPushed(any button)	*resetAlarm*	*normal*

Fig 1: Various Characterizations of a State.

We can characterize a state in various ways, as Figure 1 shows for the state ***Alarm ringing*** on a watch. The state has a suggestive name and a natural-language description of its purpose. The event sequence that leads to the state consists of setting the alarm, doing anything that doesn't clear the alarm, and then having the target time occur. A declarative condition for the state is given in terms of parameters, such as *current* and *target time*; the alarm stops ringing after 20 seconds. Finally, a stimulus-response table shows the effect of events *current time* and *button pushed*, including the response that occurs and the next state. The different descriptions of a state may overlap.

Transitions & Conditions:

- A transition is an instantaneous change from one state to another.
- The transition is said to fire upon the change from the source state to target state.
- The origin and target of a transition usually are different states, but sometimes may be the same.
- A transition fires when its events (multiple objects) occurs but only if the guard condition is true. For example, "when you go out in the morning (event), if the temperature is below freezing(condition), then put on your gloves (next state).".
- 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.

Guard condition Vs. change event:

Guard condition	change event
A Guard condition is checked only once	A Change event is checked continuously
UML notation for a transition is a line followed by guard condition in square brackets	May include event label in italics and **when** keyword. From the origin state to the target state an arrowhead points to the target state.

Example:

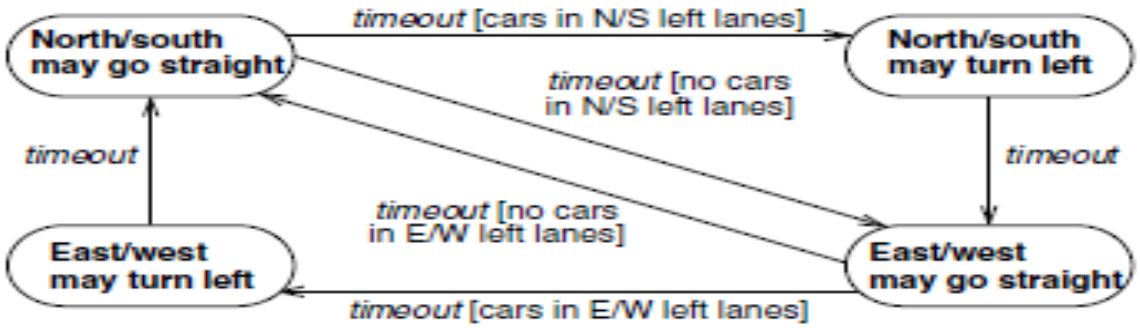


Figure 5.7 Guarded transitions. A transition is an instantaneous change from one state to another. A guard condition is a boolean expression that must be true in order for a transition to occur.

Figure 5.7 shows guarded transitions for traffic lights at an intersection. One pair of electric eyes checks the north-south left turn lanes; another pair checks the east-west turn lanes. If no car is in the north-south and/or east-west turn lanes, then the traffic light control logic is smart enough to skip the left turn portion of the cycle.

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.
- State names must be unique within the scope of a state diagram.
- 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.
- State diagrams interact by passing events and through the side effects of guard conditions.
- UML notation for a state diagram is a rectangle with its name in small pentagonal tag in the upper left corner.
- The constituent states and transitions lie within the rectangle.
- States do not totally define all values of an object.
- If more than one transition leaves a state, then the first event to occur causes the corresponding transition to fire.
- If an event occurs and no transition matches it, then the event is ignored.
- If more than one transition matches an event, only one transition will fire, but the choice is nondeterministic.
- A class with more than one state has important temporal behavior. Similarly, a class is temporally important if it has a single state with multiple responses to events.

Sample State Diagram:

Figure 5.5 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. The UML notation for a state diagram is a rectangle with its name in a small pentagonal tag in the upper left corner. The constituent states and transitions lie within the rectangle.

At the start of a call, the telephone line is idle. When the phone is removed from the hook, it emits a dial tone and can accept the dialing of digits. Upon entry of a valid number, the phone system tries to connect the call and route it to the proper destination. The connection can fail if the number or trunk are busy. If the connection is successful, the called phone begins ringing. If the called party answers the phone, a conversation can occur. When the called party hangs up, the phone disconnects and reverts to idle when put on hook again.

States do not totally define all values of an object. For example, state *Dialing* includes all sequences of incomplete phone numbers. It is not necessary to distinguish between different numbers as separate states, since they all have the same behavior, but the actual number dialed must of course be saved as an attribute.

Eg: Sample State diagram

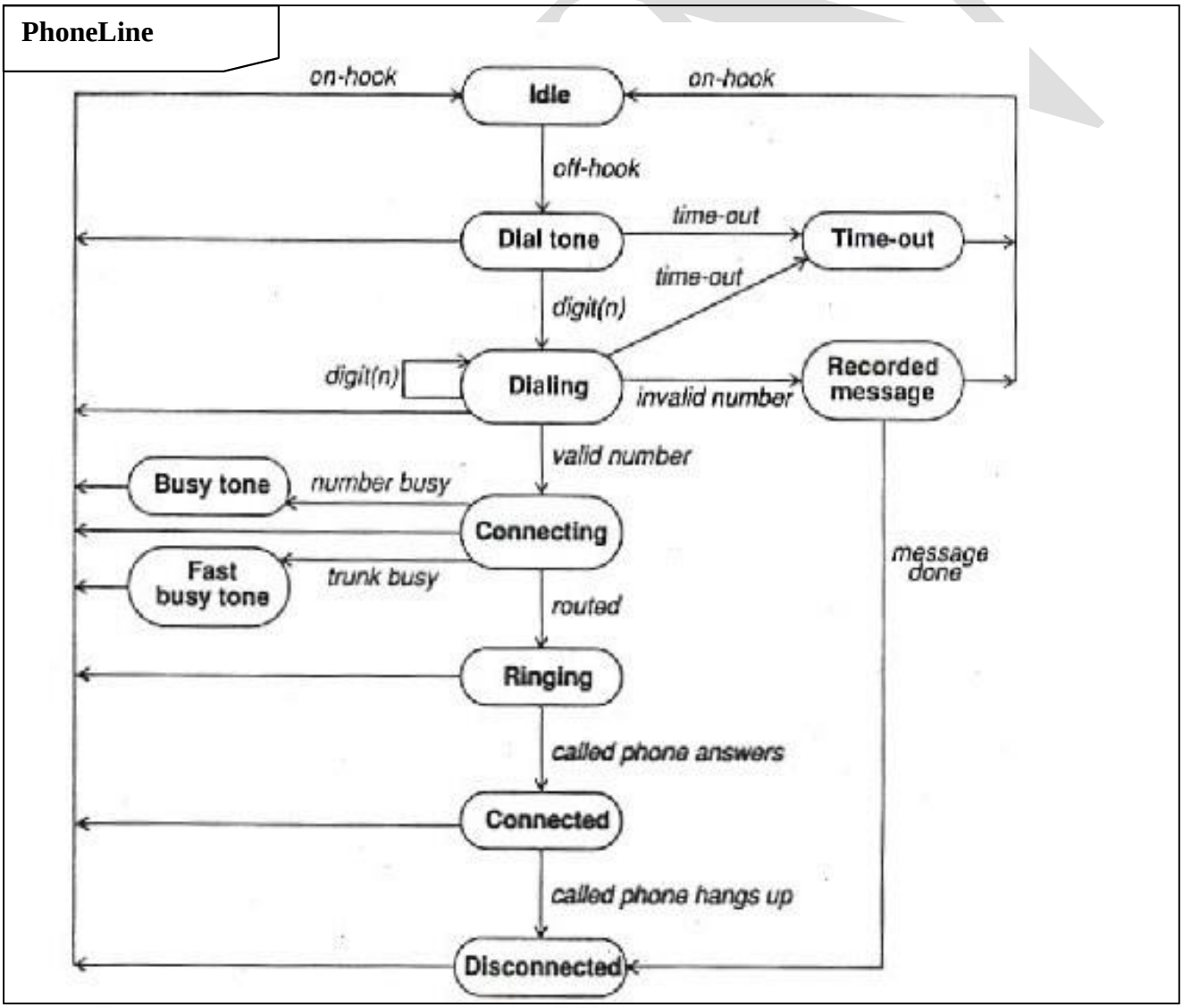


Figure 5.5 State diagram for phone line

One shot state diagram:

- State diagrams can represent continuous loops or one-shot life cycles Diagram for the Phone line is a continuous loop.
- One – shot state diagrams represent objects with finite lives and have initial and final states.
  - ➢ Initial state is entered on creation of an object
  - ➢ Entry of the final state implies destruction of the object.

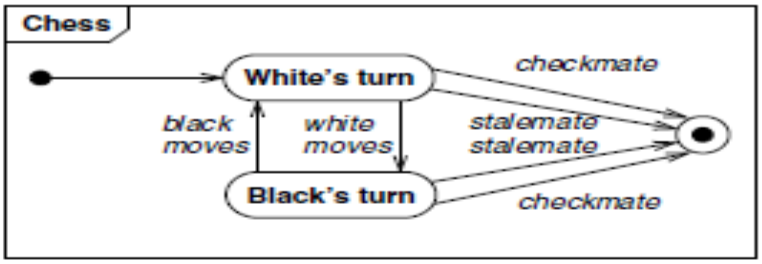


Figure 5.9 State diagram for chess game. One-shot diagrams represent objects with finite lives.

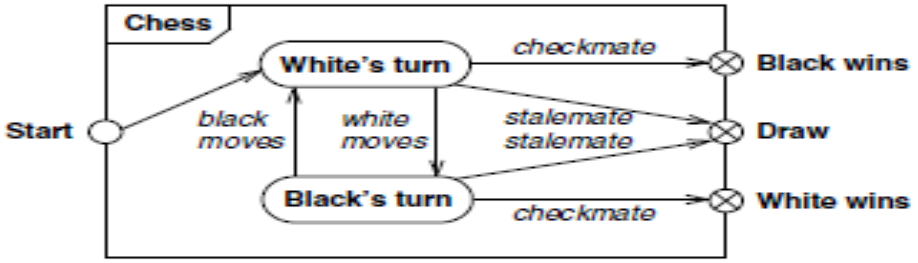


Figure 5.10 State diagram for chess game. You can also show one-shot diagrams by using entry and exit points.

5.4.3 Summary of Basic State Diagram Notation

Figure 5.11 summarizes the basic UML syntax for state diagrams.

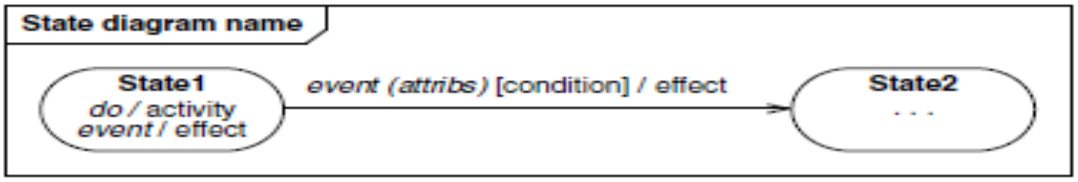


Figure 5.11 Summary of basic notation for state diagrams.

State diagram Behaviour:

1.Activity effects

- An effect is a reference to a behavior that is executed in response to an event.
- An activity is the actual behavior that can be invoked by any number of effects.
  Eg: disconnectPhoneLine might be an activity that executed in response to an onHook event for Figure5.5.
- Activities can also represent internal control operations, such as setting attributes or generating other events.
- An activity may be performed upon a transition, upon the entry to or exit from a state, or upon some other event within a state.
- The notation for an activity is a slash ("/") and the name (or description) of the activity, following the event that causes it. The keyword do is reserved for indicating an ongoing activity and may not be used as an event name.

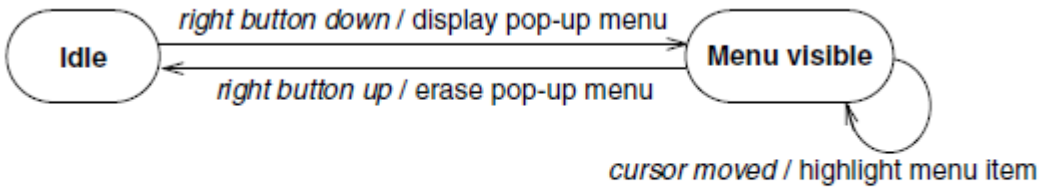


Figure 5.12 Activities for pop-up menu. An activity is behavior that can be executed in response to an event.

- Figure 5.12 shows the state diagram for a pop-up menu on a workstation. When the right button is depressed, the menu is displayed; when the right button is released, the menu is erased. While the menu is visible, the highlighted menu item is updated whenever the cursor moves.

2. Do-Activities:

- A do-activity is an activity that continues for an extended time. By definition, a do-activity can only occur within a state and cannot be attached to a transition. For example, the warning light may flash during the Paper jam state for a copy machine (Figure 5.13).

- Do-activities include continuous operations, such as displaying a picture on a television screen, as well as sequential operations that terminate by themselves after an interval of time, such as closing a valve.

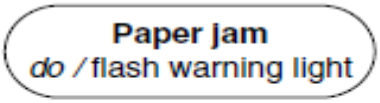


Figure 5.13 Do-activity for a copy machine. A do-activity is an activity that continues for an extended time.

- The notation "do /" denotes a do-activity that may be performed for all or part of the duration that an object is in a state.

3.Entry and Exit Activities:

- As an alternative to showing activities on transitions,bind activities to entry or to exit from a state.
- For example, Figure 5.14 shows the control of a garage door opener. The user generates *depress* events with a pushbutton to open and close the door.
- Each event reverses the direction of the door, but for safety the door must open fully before it can be closed. The control generates *motor up* and *motor down* activities for the motor.
- The motor generates *door open* and *door closed* events when the motion has been completed. Both transitions entering state *Opening* cause the door to open.

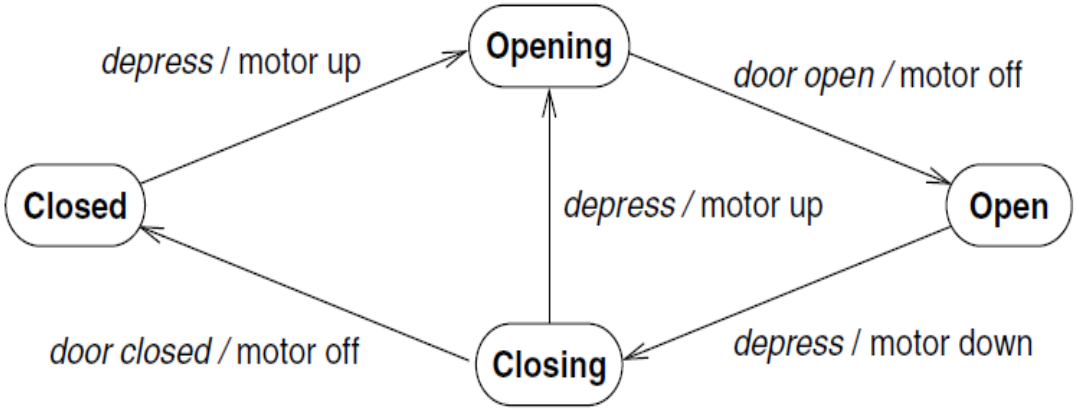


Figure 5.14 Activities on transitions. An activity may be bound to an event that causes a transition.

- Figure 5.15 shows the same model using activities on entry to states.
- An entry activity is shown inside the state box following the keyword *entry* and a "/" character. Whenever the state is entered, by any incoming transition, the entry activity is performed.
- An entry activity is equivalent to attaching the activity to every incoming transition. If an incoming transition already has an activity, its activity is performed first.

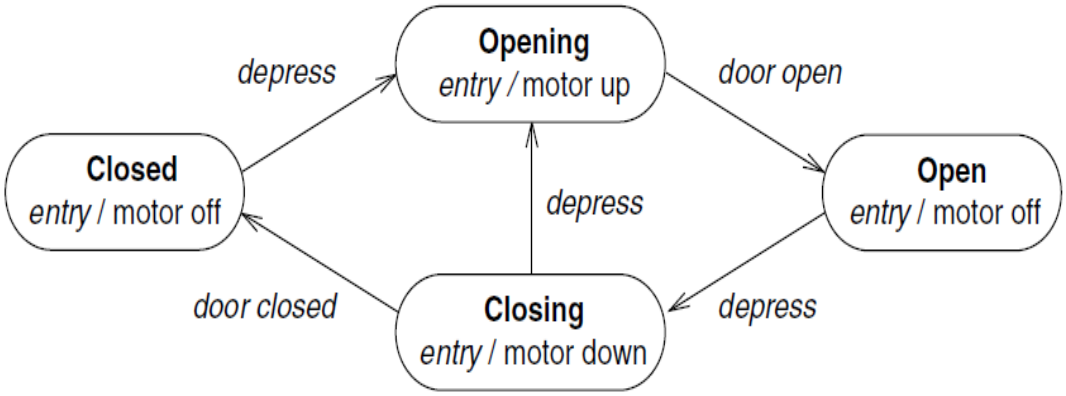


Figure 5.15 Activities on entry to states. An activity may also be bound to an event that occurs within a state.

- Exit activities are less common than entry activities, but they are occasionally useful. An exit activity is shown inside the state box following the keyword *exit* and a "/" character.
- Whenever the state is exited, by any outgoing transition, the exit activity is performed first. If a state has multiple activities, they are performed in the following order: activities on the **incoming transition, entry activities, do-activities, exit activities, activities on the outgoing transition**.
- Events that cause transitions out of the state can interrupt do-activities. If a doactivity is interrupted, the exit activity is still performed.
- In general, any event can occur within a state and cause an activity to be performed. *Entry* and *exit* are only two examples of events that can occur.

- As Figure 5.16 shows, there is a difference between an event **within a state and a self-transition**; only the self-transition causes the entry and exit activities to be executed.

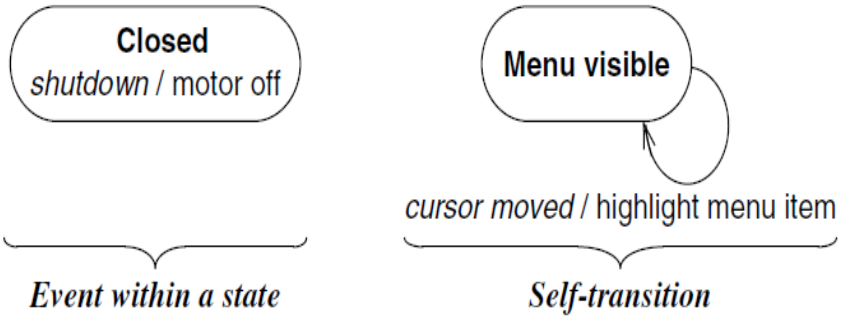


Figure 5.16 Event within a state vs. self-transition. A self-transition causes entry and exit activities to be executed. An event within a state does not.

4. Completion Transition:

- When the activity is completed, a transition to another state fires. An arrow without an event name indicates an automatic transition that fires when the activity associated with the source state is completed. Such unlabeled transitions are called ***completion transitions*** because they are triggered by the completion of activity in the source state.
- A guard condition is tested only once, when the event occurs. If a state has one or more completion transitions, but none of the guard conditions are satisfied, then the state remains active and may become "stuck"—the completion event does not occur a second time, therefore no completion transition will fire later to change the state.
- If a state has completion transitions leaving it, normally the guard conditions should cover every possible outcome. You can use the special condition *else* to apply if all the other conditions are false. Do not use a guard condition on a completion transition to model waiting for a change of value. Instead model the waiting as a change event.

5. Sending Signals:

- An object can perform the activity of sending a signal to another object.
- A system of objects interacts by exchanging signals. The activity **"send *target.S*(*attributes*)"** sends signal *S* with the given attributes to the target object or objects. For example, the phone line sends a *connect(phone number)* signal to the switcher when a complete phone number has been dialed.
- A signal can be directed at a set of objects or a single object. If the target is a set of objects, each of them receives a separate copy of the signal concurrently, and each of them independently processes the signal and determines whether to fire a transition.
- If the signal is always directed to the same object, the diagram can omit the target (but it must be supplied eventually in an implementation).
- Beware of unwanted race conditions in state diagrams. **Race conditions** may occur when a state can accept events from more than one object.

6. *Sample State Diagram with Activities:*

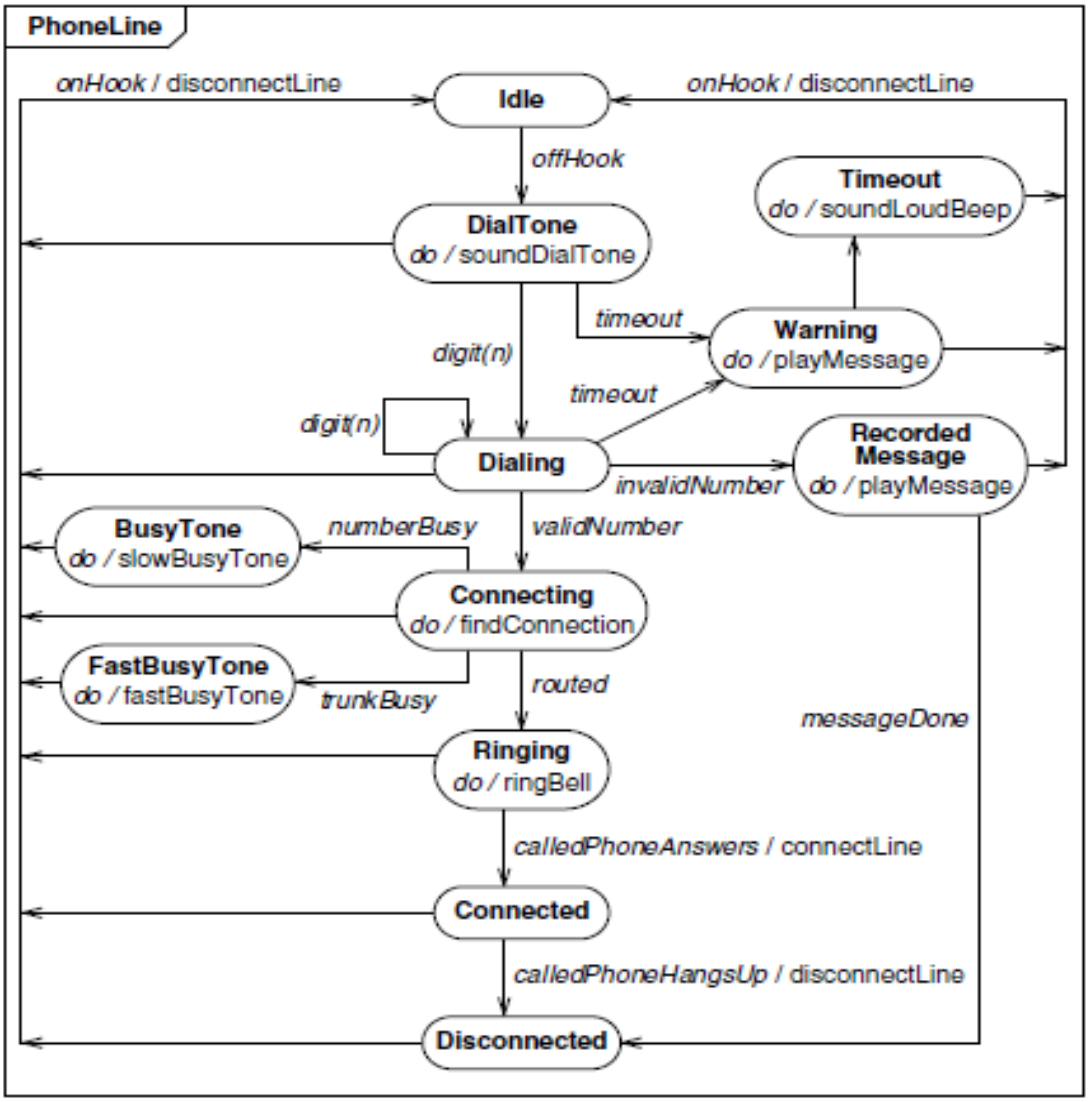


Figure 5.17 State diagram for phone line with activities. State diagrams let you express what objects do in response to events.

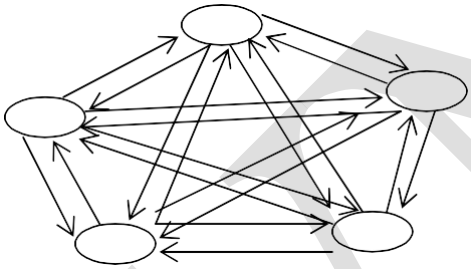
Advanced State Diagrams

- **Nested state diagram**
- **Nested states**

Problem with Flat state diagrams:

- Flat unstructured state diagrams are impractical for large problems, because – representing an object with 'n' independent Boolean attribute requires 2^n states.
- By partitioning the state into 'n' independent state diagram requires 2n states only.

Eg:



Above figure requires 'n^2' transition to connect every state to other state. This can be reduced to as low as n by using sub diagrams structure.

Expanding states

- One way to organize a model is by having high level diagram with sub diagrams expanding certain state. This is like a macro substitution in programming language.
- Figure 6.2 shows such a state diagram for a vending machine. Initially, the vending machine is idle. When a person inserts coins, the machine adds the amount to the cumulative balance. After adding some coins, a person can select an item. If the item is empty or the balance is insufficient, the machine waits for another selection. Otherwise, the machine dispenses the item and returns the appropriate change.

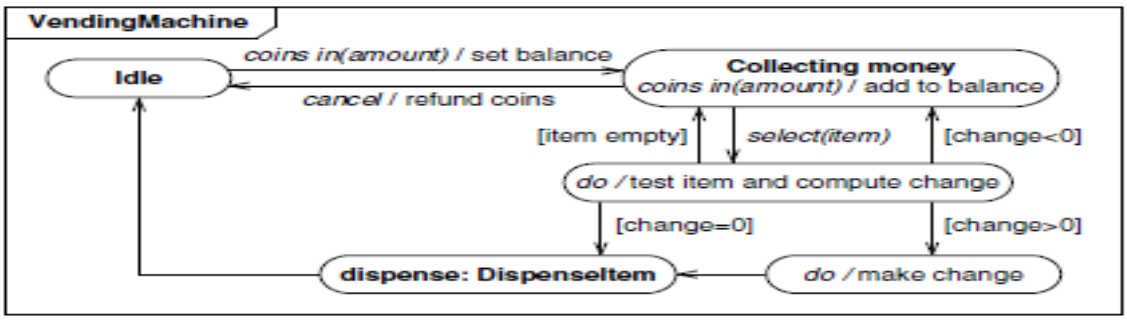


Figure 6.2 Vending machine state diagram. You can simplify state diagrams by using subdiagrams.

- Figure 6.3 elaborates the *dispense* state with a lower-level state diagram called a submachine.
- A ***submachine*** is a state diagram that may be invoked as part of another state diagram.
- The UML notation for invoking a submachine is to list a local state name followed by a colon and the submachine name.
- Conceptually, the submachine state diagram replaces the local state. Effectively, a submachine is a state diagram "subroutine."

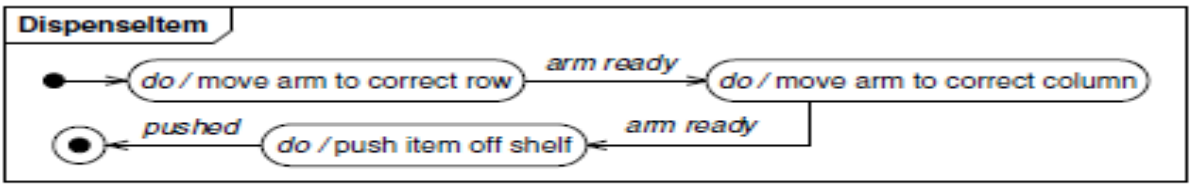


Figure 6.3 ***Dispense item*** **submachine of vending machine**. A lower-level state diagram can elaborate a state.

Nested States:

- Figure 6.4 simplifies the phone line model a single transition from *Active* to *Idle* replaces the transitions from each state to *Idle.*
- All the original states except *Idle* are nested states of *Active.*
- The occurrence of event *onHook* in any nested state causes a transition to state *Idle*.
- The ***composite state*** name labels the outer contour that entirely encloses the nested states.
- Thus, *Active* is a composite state with regard to nested states *DialTone*, *Timeout*, *Dialing*, and so forth.
- Nest states to an arbitrary depth. A nested state receives the outgoing transitions of its composite state.

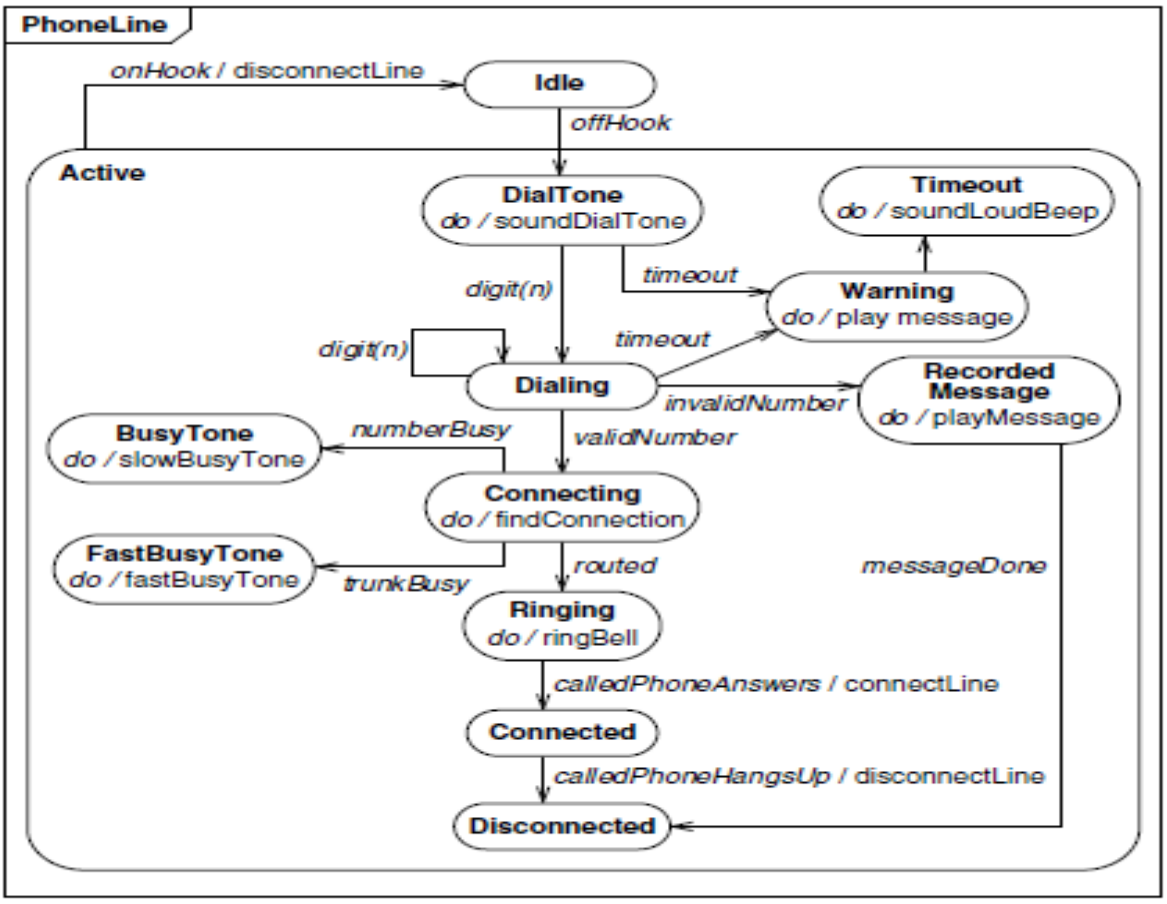


Figure 6.4 Nested states for a phone line. A nested state receives the outgoing transitions of its enclosing state.

- Figure 6.5 shows a state diagram for an automobile automatic transmission.
- The transmission can be in reverse, neutral, or forward; if it is in forward, it can be in first, second, or third gear. States *First, Second,* and *Third* are nested states of state *Forward*.
- Each of the nested states receives the outgoing transitions of its composite state. Selecting "N" in any forward gear shifts a transition to neutral.
- The transition from *Forward* to *Neutral* implies three transitions, one from each forward gear to neutral. Selecting "F" in neutral causes a transition to forward. Within state *Forward*, nested state *First* is the default initial state, shown by the unlabeled transition from the solid circle within the *Forward* contour. *Forward* is just an abstract state; control must be in a real state, such as *First.*
- All three nested states share the transition on event *stop* from the *Forward* contour to state *First*. In any forward gear, stopping the car causes a transition to *First*.

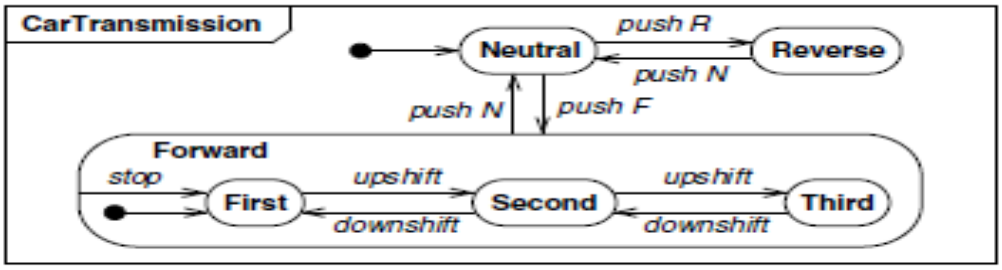


Figure 6.5 Nested states. You can nest states to an arbitrary depth.

- Entry and exit activities are particularly useful in nested state diagrams because they permit a state (possibly an entire subdiagram) to be expressed in terms of matched entry-exit activities without regard for what happens before or after the state is active.
- Transitioning into or out of a nested state can cause execution of several entry or exit activities, if the transition reaches across several levels of nesting. The entry activities are executed from the outside in and the exit activities from the inside out. This permits behavior similar to nested subroutine calls.

**********END**********