

State Modeling

Module 1-Chapter 2

Introduction

- A state model describes the sequences of operations that occur in response to external stimuli.
- As opposed to what the operations do, what they operate on, or how they are implemented.
- Changes to objects and their relationships over time should be examined.
- A state model consists of multiple state diagrams, one for each class with temporal behavior that is important to an application.
- A state diagram relates **events and states**.
- Events represent external stimuli
- States represent values of (attributes) objects ٧

Events (1)

- An event is an occurrence at a point of time ∩
- Examples:
 - User depresses left button
 - Flight 123 departs from Amman
 - Power turned on
 - Alarm set
 - Paper tray becomes empty
- Events corresponds to verb in the past tense or the onset of some condition.
- By definition an event happens instantaneously. The time at which an event occurs is an implicit attribute of the event.

Events (2)

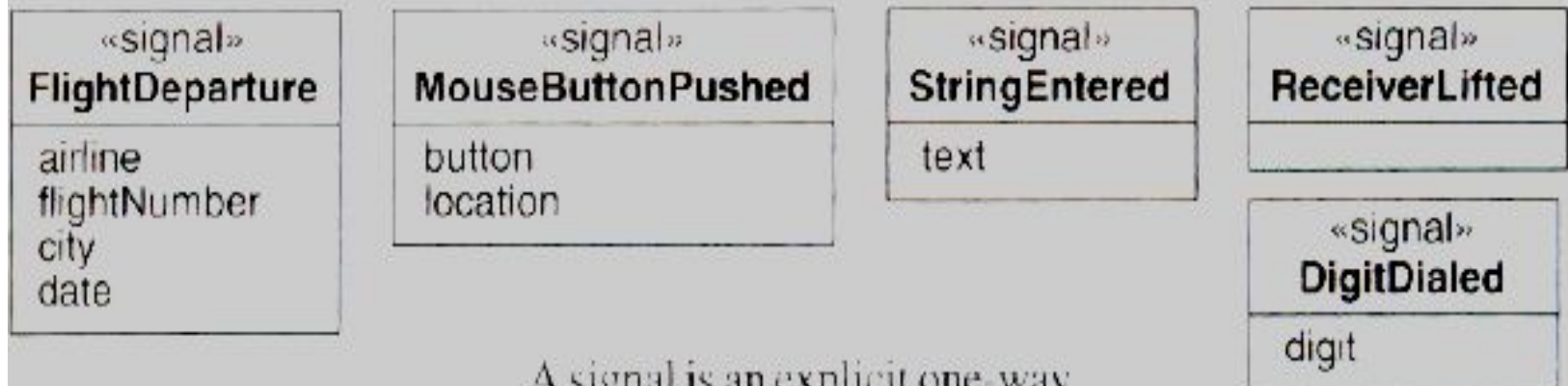
- One event may logically precede or follow another, or the two events are unrelated. Events that are causally unrelated are said to be concurrent.
- Events include error conditions as well as normal occurrences.
- Examples of error events:
 - » Motor jammed
 - » Transaction aborted
 - » Timeout
- There are several kinds of events. The most common are:
 - » Signal event
 - » Change event
 - » Time event

Signal Events (1)

- A signal is a one-way transmission of information from one object to another object.
 - It is different from a function 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 to the second object, which may or may not choose to send it.
-
- A signal event is the event of sending or receiving a signal.
 - In general we are more concerned about the receipt of a signal because it causes effects in the receiving object.
 - **Difference** between signal and **signal event**: the first one is a message between objects, the second one is an occurrence in time.

- Every signal transmission is a unique occurrence, but they can be grouped into signal classes and give each signal class a name to indicate common structure and behavior.
- E.g. *UAflight 123 departs from Chicago on January 10, 1991* is an instance of signal class *FlightDeparture*.
- Some signals are simple occurrences, but most signal classes have attributes indicating the values they convey.
- Flight Departure has attributes airline, flightNumber, city, and date.
- The UML notation is the keyword **signal** in **guillemets** (« ») above the signal class name in the top section of a box. The second section lists the signal attributes.

Signal classes and attributes.



A signal is an explicit one-way transmission of information from one object to another.

Change Event

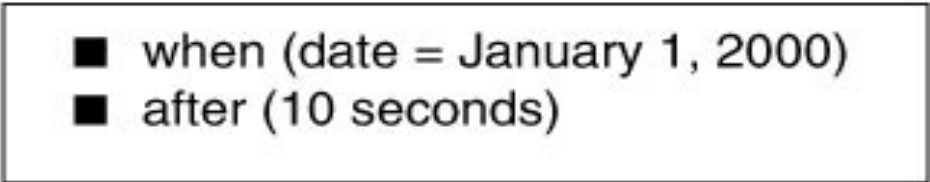
- A change event is an event that is caused by the satisfaction of a boolean expression.
- The intent of a change event is that the expression is continually tested and whenever the expression changes from false to true the event happens. (an implementation would not continuously check a change event)
- The **UML notation** for a change event is the keyword **when** followed by a **parenthesized Boolean expression**.
- Examples:

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

Figure 5.2 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 tie 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.
- Examples:



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

Figure 5.3 Time events. A time event is an event caused by the occurrence of an absolute time or the elapse of a time interval.

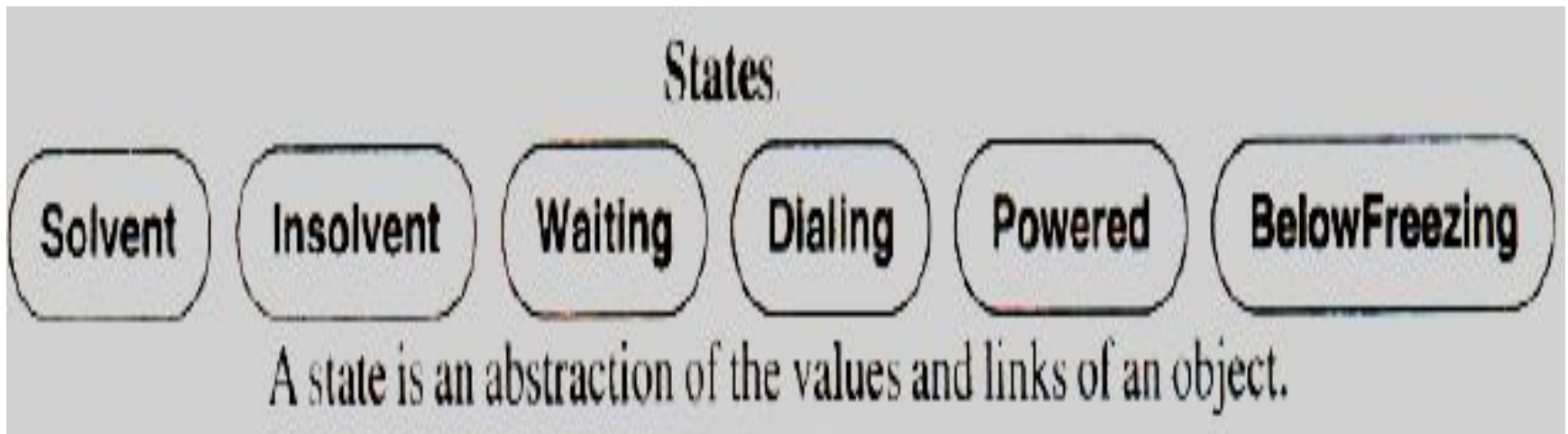
Object-Oriented Modeling and Design with UML, Second Edition
by Michael Blaha and James Rumbaugh. ISBN 0-13-1-01592-0-4.
© 2005 Pearson Education, Inc., Upper Saddle River, NJ.
All rights reserved.

States

- A state is an abstraction of the values and the links of an object.
- Sets of values and links are grouped together into a state according to the gross behavior of objects.
- Example: The state of an bank is either solvent or insolvent, depending on whether its assets exceed its liabilities.
- Ex: State of telephone :waiting , dialing
- States often correspond to:
 - verbs with a suffix 'ing': waiting, dialing..
 - Or the duration of some condition: powered, below freezing.

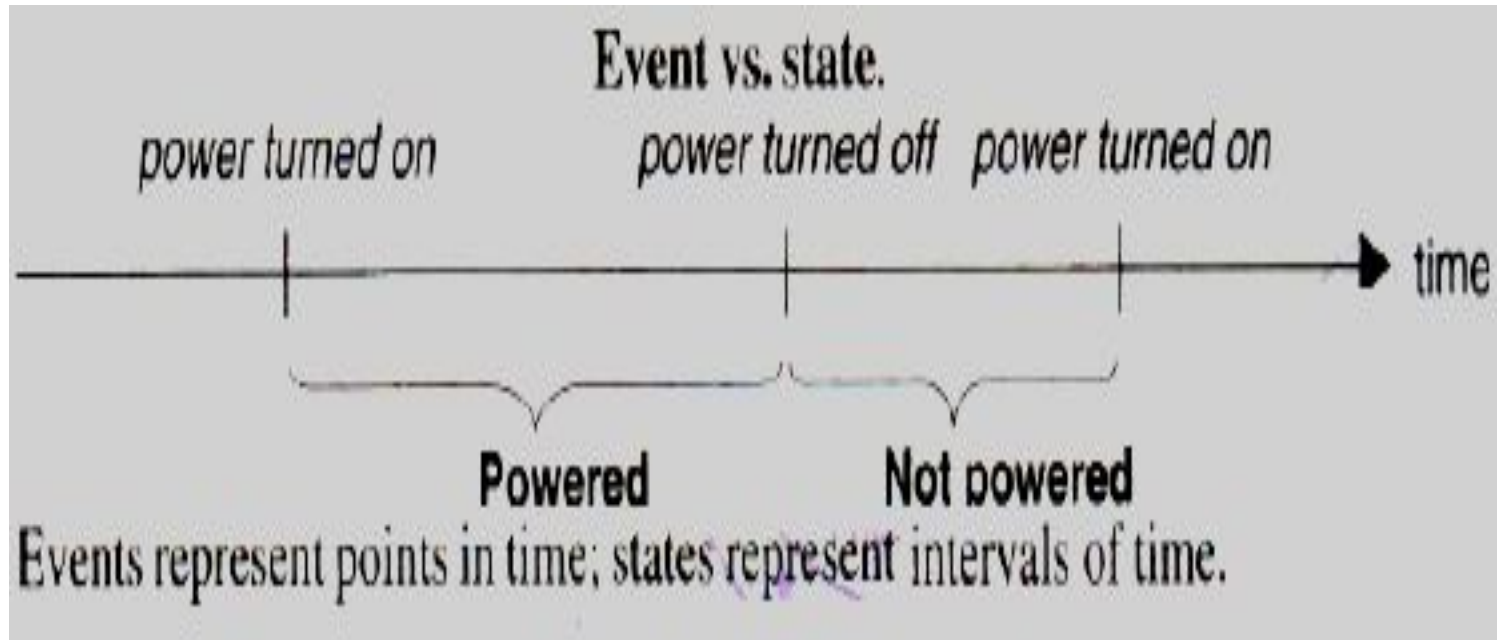
UML notation for a state:

- a **rounded box** containing an **optional state name**. Convention is to list the state name in **boldface**, center the name near the top of the box, and **capitalize** the first letter.



- 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.
- The response may include the invocation of behavior or a change of state.
- E.g. If a digit is dialed in state **Dialtone**, the phone line drops the dial tone and enters state **Dialing**; if the receiver is replaced in state **Dialtone**, the phone line goes dead and enters state **Idle**.

- There is certain symmetry between events and states as figure illustrates.



- Events represent points in time; states represent intervals of time. A state corresponds to the interval between two events received by an object.

- E.g. After the receiver is lifted and before the first digit is dialed, the phone line is in state ***Dialtone***.
- The state of an object depends on *past events*, which in most cases are eventually *hidden by subsequent events*. E.g. Events that happened before the phone is hung up do not affect future behavior; the ***Idle*** state "forgets" events received prior to the receipt of the hang up signal.

- Like objects *links can have state*. As a practical matter, it is generally sufficient to associate state *only with objects*.

State

- States may be characterized in various ways.
- Here one way to characterize states:

State: *AlarmRinging*

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 characterizes the state:

alarm = on, alarm set to *targetTime*, $targetTime \leq currentTime \leq targetTime + 20$ seconds, and no button has been pushed since *targetTime*

Events accepted in the state:

event	response	next state
when (<i>currentTime = targetTime + 20</i>)	<i>resetAlarm</i>	<i>normal</i>
<i>buttonPushed</i> (any button)	<i>resetAlarm</i>	<i>normal</i>

Figure 5.6 Various characterizations of a state. A state specifies the response of an object to input events.

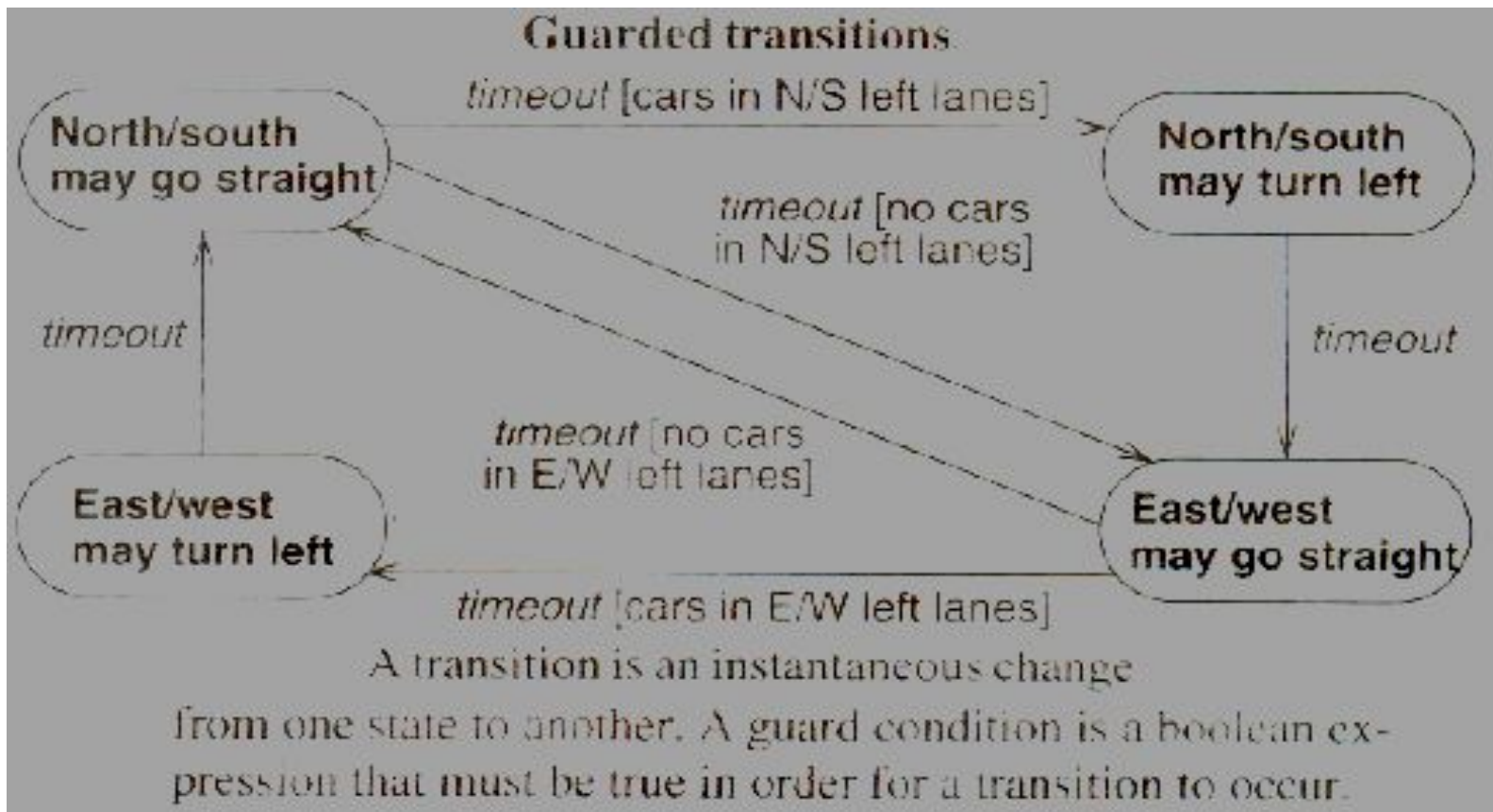
Transitions and Conditions

- 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 the target state. Usually the origin and target states are different but may be the same.
- A transition occurs when its event occurs, unless an optional guard condition causes the event to be ignored.
- A guard condition is a boolean expression that must be true in order for a transition to occur.
- The choice of next state depends on both the source state and the event received.

Guard condition:

- is a Boolean expression that must be true in order for a transition to occur.
- E.g. A traffic light at an intersection may change only if a road has cars waiting.
- A guarded transition *fires* when its event occurs, *but only if the guard condition is true*.
- E.g. "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 *checked only once, at the time the event occurs*, and the transition fires if the condition is true. If the condition becomes true later, the transition *does not then fire*. Note that a guard condition is *different from a change event* - a *guard condition is checked only once* while a *change event is, in effect, checked continuously*.

- Figure 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.



UML notation for a transition:

- is a line from the origin state to the target state. An arrowhead points to the target state. The line may consist of several line segments. An event may label the transition and be *followed by an optional guard condition* in square brackets. By convention, line segments are usually confined to a rectilinear grid. The event name is *italicized* and the condition is shown in normal font.

State Diagram

- A state diagram is a graph whose nodes are states and whose directed arcs are transitions between states.
- A state diagram specifies or describes the state sequence caused by event sequences.
- State names must be unique within the scope of the state diagram.
- All objects in a class execute the state diagram for that class.
- The class state diagram models the common behavior of the class objects.

State diagram example

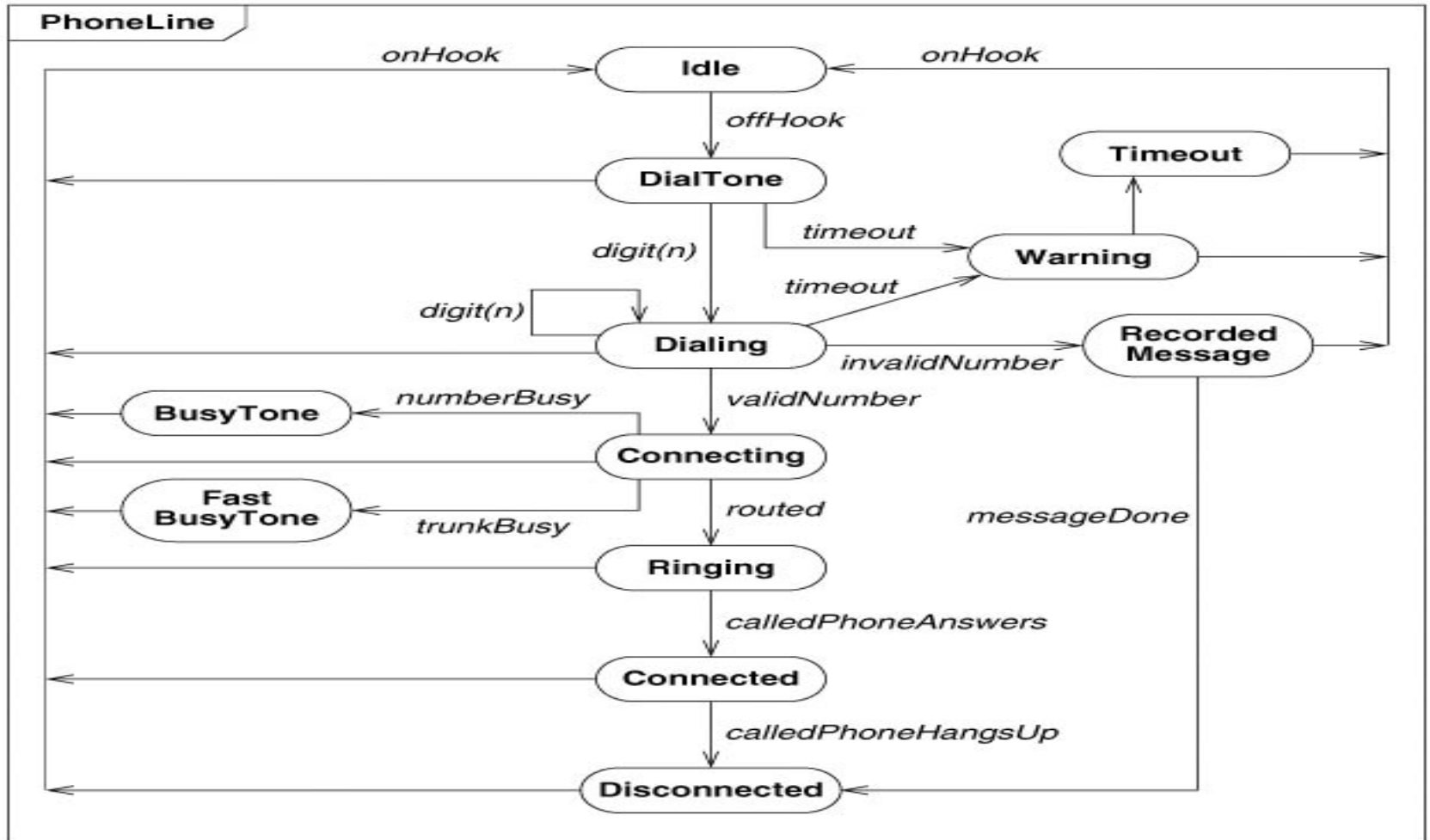


Figure 5.8 State diagram for a telephone line. A state diagram specifies the state sequences caused by event sequences.

- 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.

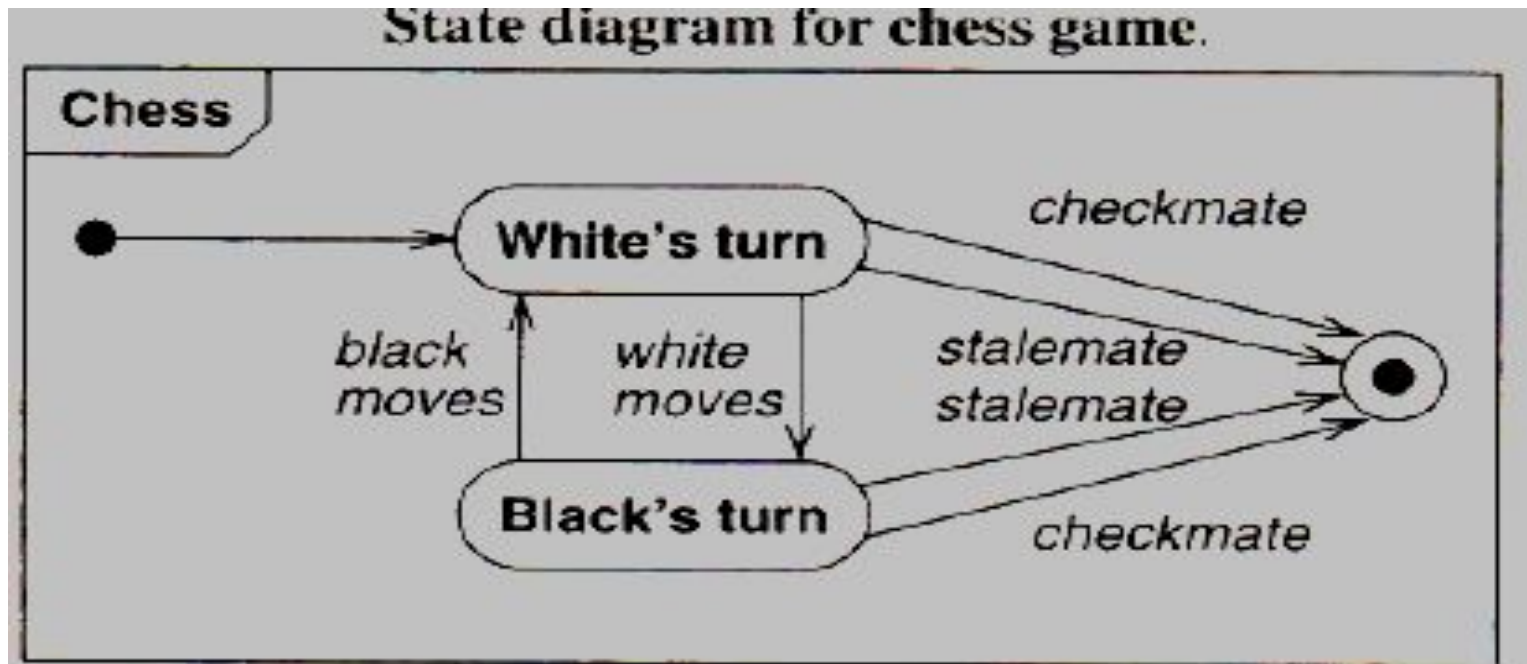
- At the start of a call, the telephone line is idle. When the phone is removed from the hook, it emits a dial tone and can accept the dialing of digits. Upon entry of a valid number, the phone system tries to connect the call and route it to the proper destination. The connection can fail if the number or trunk is 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.
- Note that the receipt of the signal ***onHook*** causes a transition from any state to ***Idle*** (the bundle of transitions leading to Idle).

- States do not totally define all values of an object. E.g. 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.
- 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*.
-

One-shot State Diagrams

- State diagrams can represent continuous loops or one-shot life cycles. The diagram for the phone line is a continuous loop. In describing ordinary usage of the phone, one does not know or care how the loop is started. (While describing installation of new lines, the initial state would be important.)

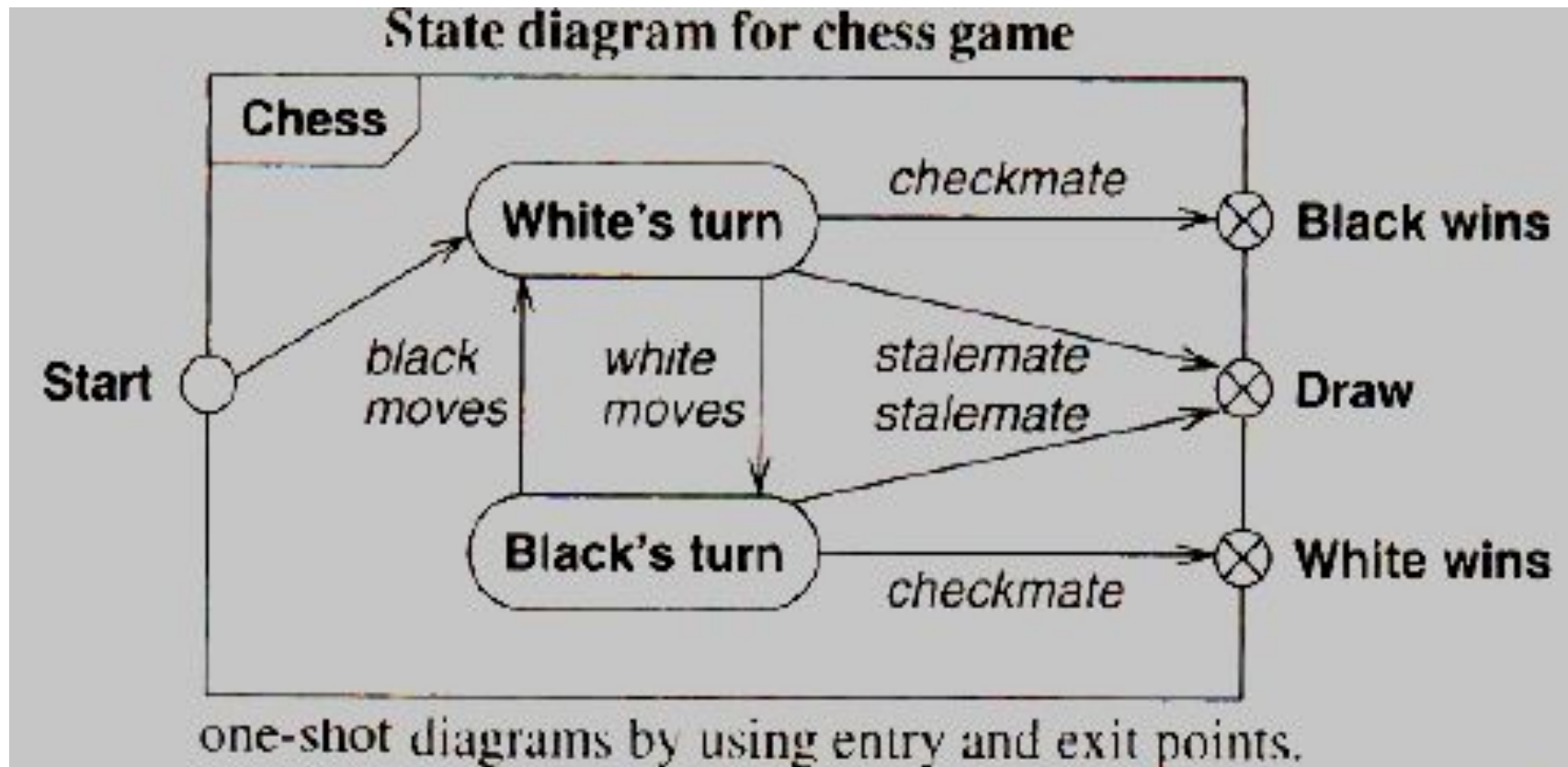
- One-shot state diagrams represent objects with finite lives and have initial and final states. The initial state is entered on *creation* of an object; entry of the final state implies *destruction* of the object.
- UML notation for a state diagram , Figure shows a simplified life cycle of a chess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



One-shot diagrams represent objects with finite lives.

As an alternate notation, initial and final states can be indicated 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.



State Diagram Behavior:

- State diagrams would be of little use if they just described events. A full description of an object must specify *what the object does in response to events*.

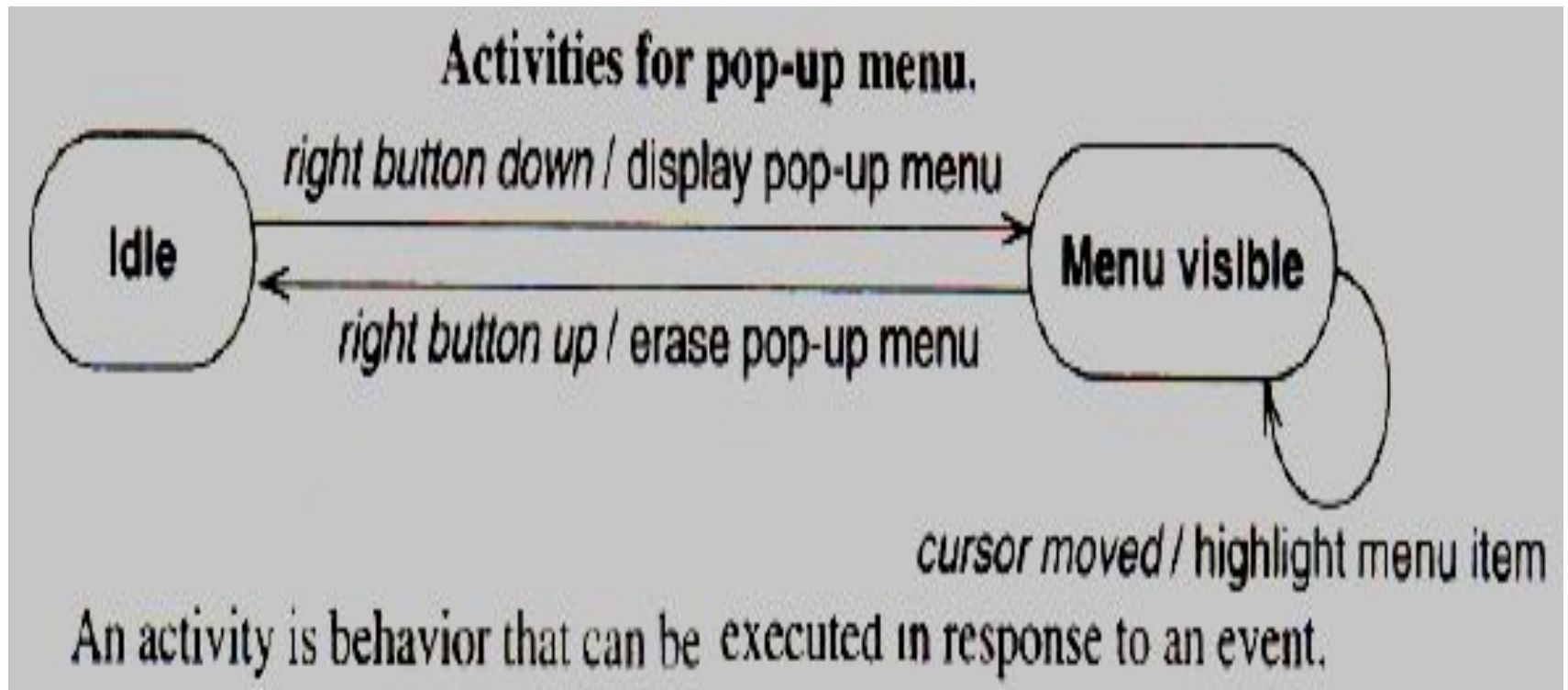
Activity Effects:

- 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. E.g. *disconnectPhoneLine* might be an activity that is executed in response to an *onHook* event.

- 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. Activities can also represent internal control operations, such as setting attributes or generating other events.
- Such activities have no real-world counterparts but instead are mechanisms for structuring control within an implementation.
- E.g. A program might increment an internal counter every time a particular event occurs.

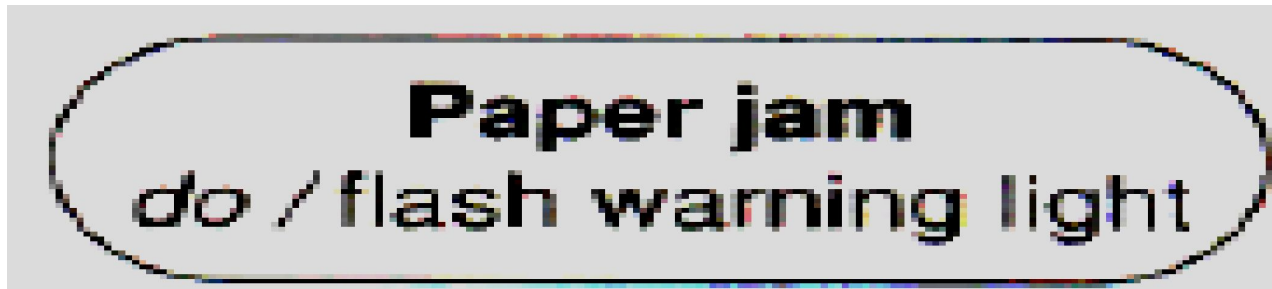
- The notation for an activity is a slash (“ /”) and the *name* (or description) of the activity, following the *event* that causes it. The keyword *do* is reserved for indicating an ongoing activity (to be explained) and **may not be used as an event name.**

- Figure 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.



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.
- E.g. Warning light may flash during the *Paperjam* state for a copy machine (Figure below).



- 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.
- 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.
- A do-activity may be interrupted by an event that is received during its execution; such an event may or may not cause a transition out of the state containing the do-activity.
- E.g. a robot moving a part may encounter resistance, causing it to cease moving.

Entry and Exit Activities:

- As an alternative to showing activities on transitions, one can bind activities to entry or to exit from a state. There is no difference in expressive power between the two notations, but frequently all transitions into a state perform the same activity, in which case it is more concise to attach the activity to the state.

E.g. Figure below 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.

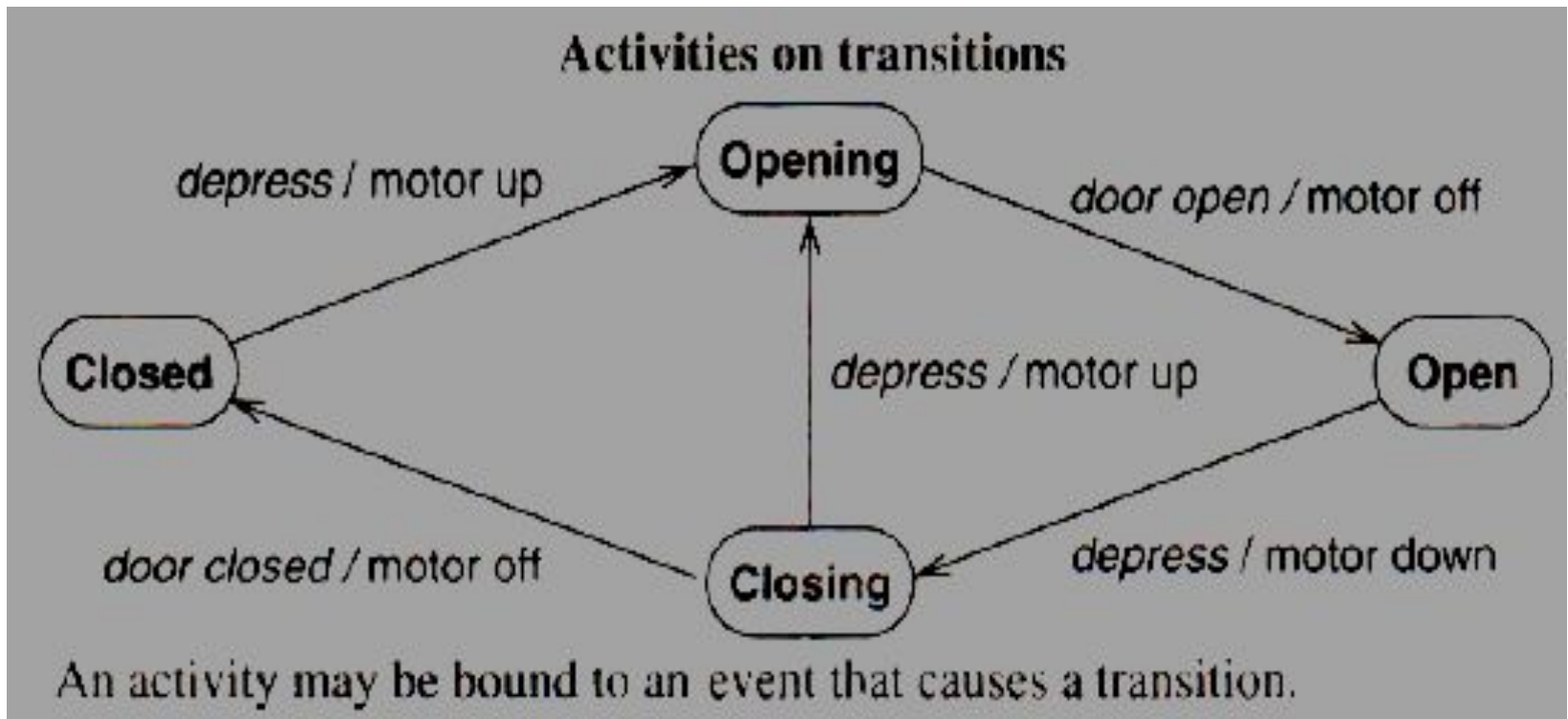
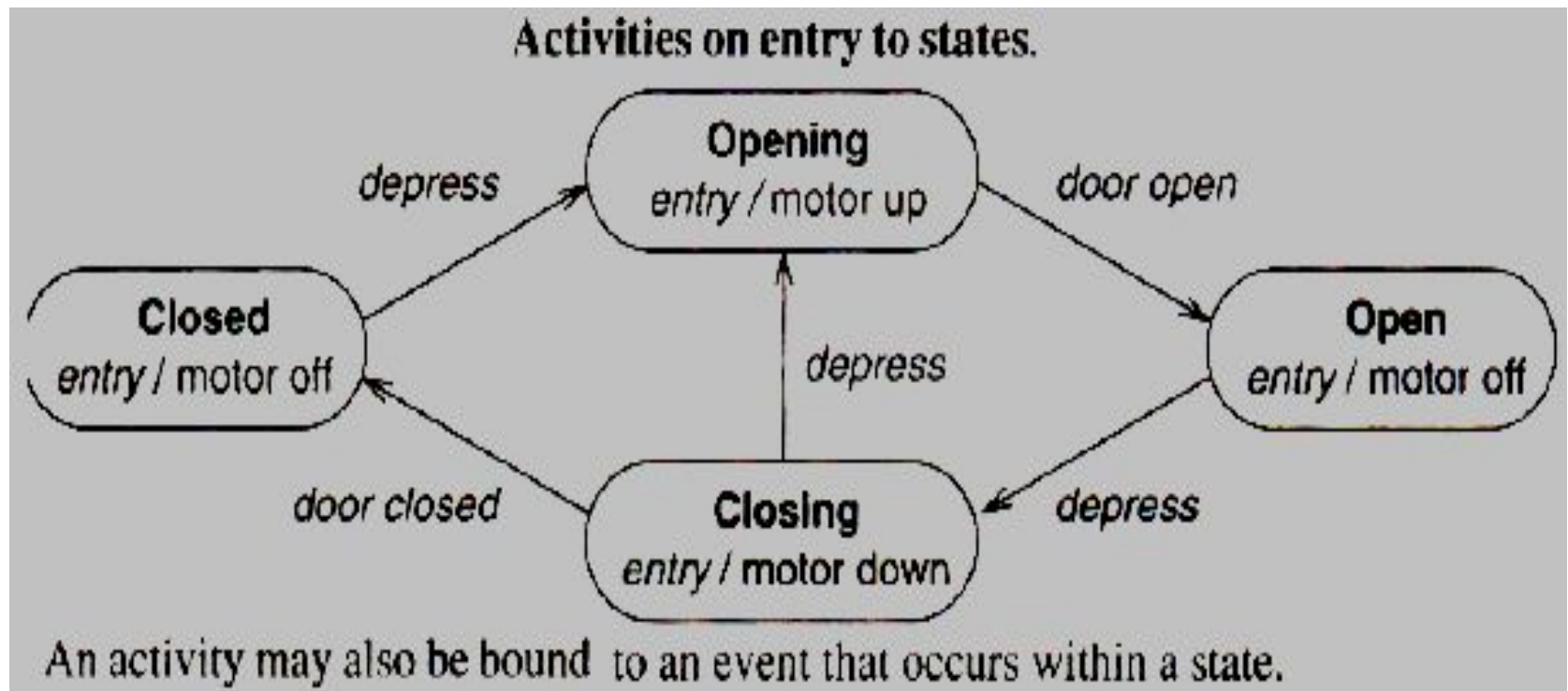


Figure below shows the same model using activities on entry to states.

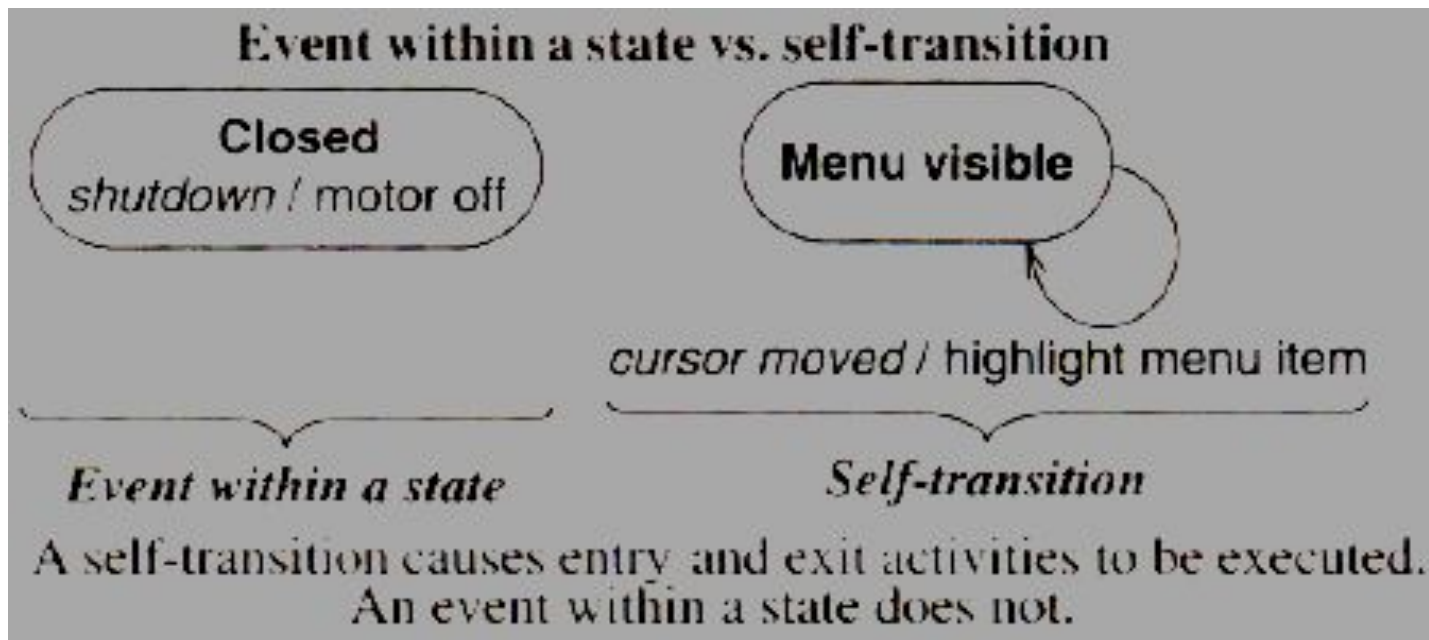
- An **entry activity** is shown inside the state box following the keyword **entry** and a "/" character. Whenever the state is entered, by any incoming transition, the entry activity is performed. An entry activity is equivalent to attaching the activity to every incoming transition. If an incoming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



- 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 do activity 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 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.



Completion Transition:

- Often the sole purpose of a state is to perform a sequential activity. 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.

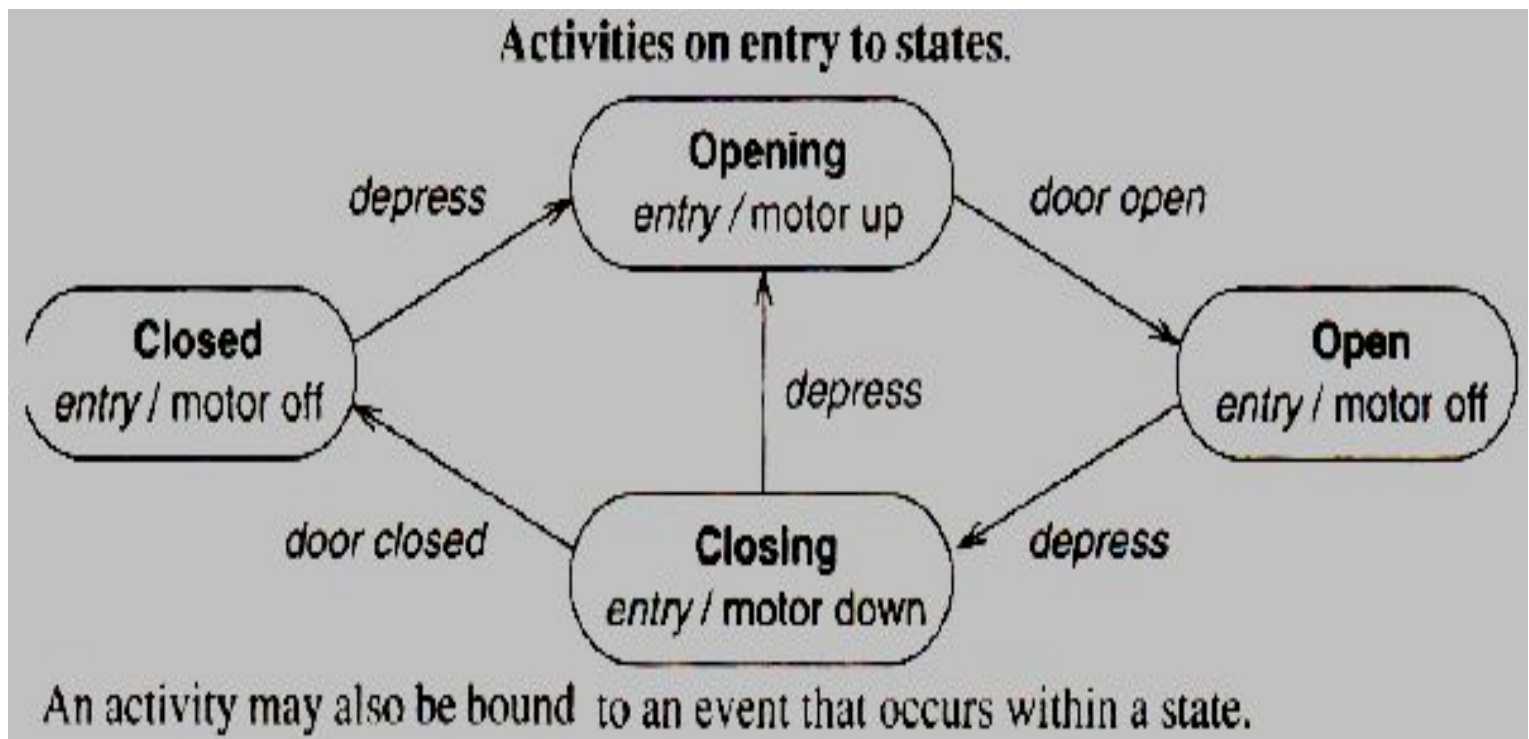
- A **guard condition** is **tested only once**, when the event occurs.
- If a state has one or more completion transitions, but none of the guard conditions are satisfied, then the state remains active and may become "stuck" - the completion event does not occur a second time, therefore no completion 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.

- The special condition *else* can be used 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.

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.
- E.g. 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, of course).

- If an object can receive signals from more than one object, the order in which concurrent signals are received may affect the final state; this is called a race condition.
- E.g. In figure below, the door may or may not remain open, if the button is pressed at about the time the door becomes fully open.



- A race condition is not necessarily a design error, but concurrent systems frequently contain unwanted race conditions that must be **avoided** by careful design.
- *A requirement of two signals being received simultaneously is never a meaningful condition* in the real world, as slight variations in transmission speed are inherent in any distributed system.

- **State:** Drawn as a rounded box containing an optional name. A special notation is available for initial states (a solid circle) and final states (a bull's-eye or encircled "x").
- **Transition:** Drawn as a line from the origin state to the target state. An arrowhead points to the target state. The line may consist of several line segments.

- **Event:** A **signal event** is shown as a label on a transition and may be followed by parenthesized attributes.
- A **change event** is shown with the keyword when followed by a parenthesized boolean expression.
- A **time event** is shown with the keyword order when followed, by a parenthesized expression involving time or the keyword after followed by a parenthesized expression that evaluates to a time duration.

- **Guard condition:** Optionally listed in square brackets after an event. Can be attached to a finalisation or state and are listed after a slash (/). Multiple effects are separated with a comma and are performed concurrently.