STATE MACHINES

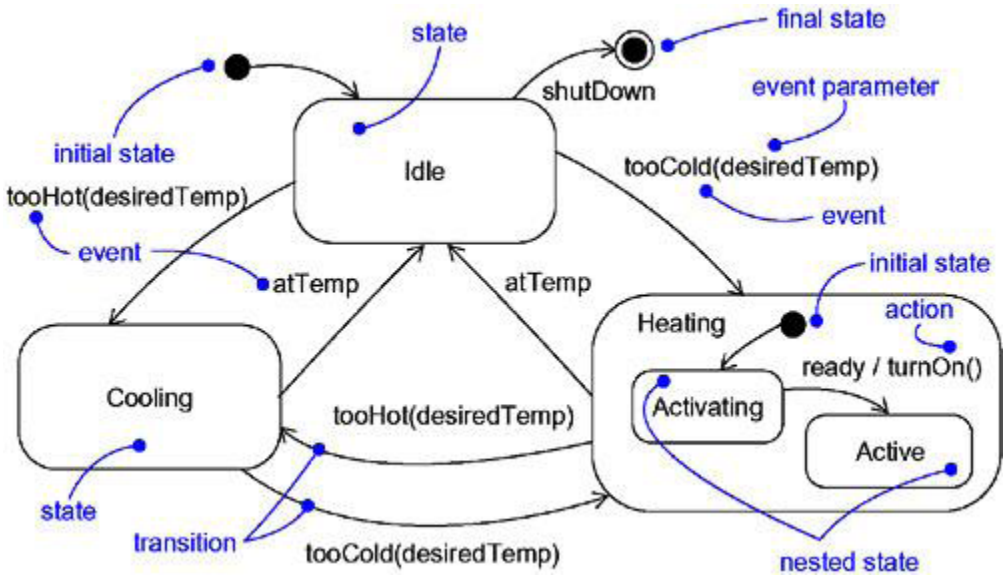


Figure 1: State Machines

state machine

- A state machine is a behavior that specifies the sequences of states an object goes through during its lifetime in response to events.
- Graphically, a state is rendered as a rectangle with rounded corners. A transition is rendered as a solid directed line.
- Figure 1 shows State Machines
- state machine are used to specif the behavior of objects that must respond to asynchronous stimulus or whose current behavior depends on their past.
- state machines are used to model the behavior of entire systems, especially reactive systems, which must respond to signals from actors outside the system.

States

- A state is a condition or situation during the life of an object during which it satisfies some condition, performs some activity, or waits for some event.
- An object remains in a state for a finite amount of time. For example, a Heater in a home might be in any of four states: Idle, Activating, Active, and ShuttingDown.

- a state name must be unique within its enclosing state
- A state has five parts: *Name, Entry/exit actions, Internal transitions* – Transitions that are handled without causing a change in state, *Substates* – nested structure of a state, involving disjoint (sequentially active) or concurrent (concurrently active) substates, *Deferred events* – A list of events that are not handled in that state but, rather, are postponed and queued for handling by the object in another state
- Figure 2 shows States
- initial state indicates the default starting place for the state machine or substate and is represented as a filled black circle
- final state indicates that the execution of the state machine or the enclosing state has been completed and is represented as a filled black circle surrounded by an unfilled circle
- Initial and final states are pseudo-states

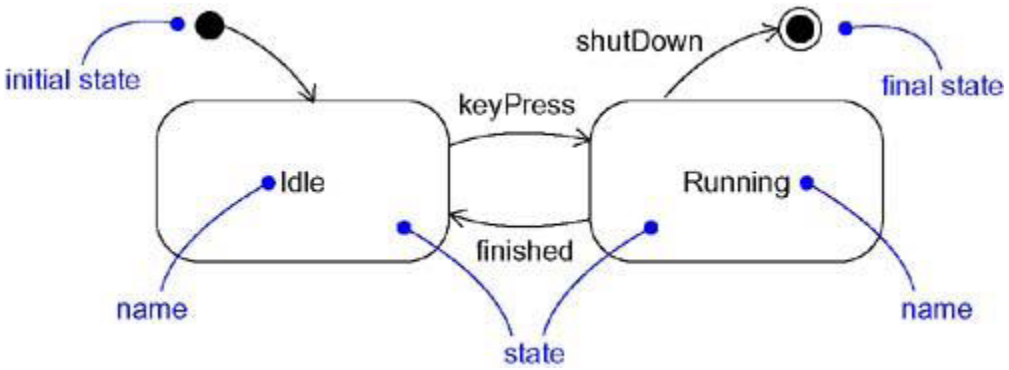


Figure 2: States

Transitions

- A transition is a relationship between two states indicating that an object in the first state will perform certain actions and enter the second state when a specified event occurs and specified conditions are satisfied.
- Transition fires means change of state occurs. Until transition fires, the object is in the source state; after it fires, it is said to be in the target state.
- A transition has five parts: *Source state* – The state affected by the transition, *Event trigger* – a stimulus that can trigger a source state to fire on satisfying guard condition, *Guard condition* – Boolean expression that is evaluated when the transition is

triggered by the reception of the event trigger, ***Action*** – An executable atomic computation that may directly act on the object that owns the state machine, and indirectly on other objects that are visible to the object, ***Target state*** – The state that is active after the completion of the transition.

- Figure 3 shows transitions
- A transition may have multiple sources as well as multiple targets
- A *self-transition*is a transition whose source and target states are the same

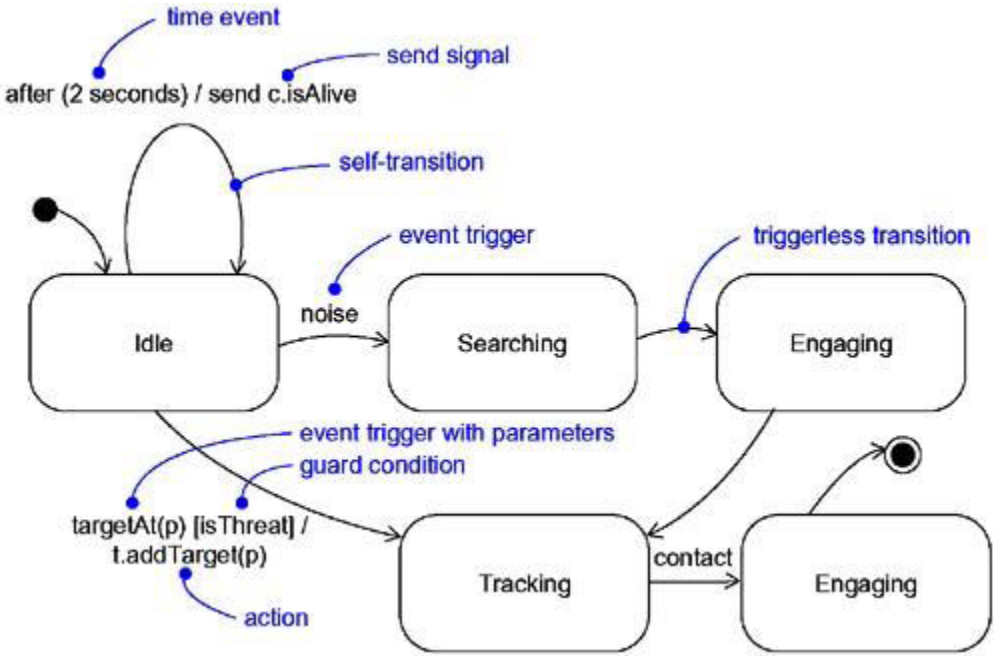


Figure 3:Transitions

Event Trigger

- An event in the context of state machines is an occurrence of a stimulus that can trigger a state transition.
- events may include signals, calls, the passing of time,or a change in state.
- An event – signal or a call – may have parameters whose values are available to the transition, including expressions for the guard condition and action.
- An event trigger may be polymorphic

Guard condition

- a guard condition is rendered as a Boolean expression enclosed in square brackets and placed after the trigger event

- A guard condition is evaluated only after the trigger event for its transition occurs
- A guard condition is evaluated just once for each transition at the time the event occurs, but it may be evaluated again if the transition is retriggered

Action

- An action is an executable atomic computation i.e, it cannot be interrupted by an event and runs to completion.
- Actions may include operation calls, the creation or destruction of another object, or the sending of a signal to an object

An activity may be interrupted by other events.

Advanced States and Transitions

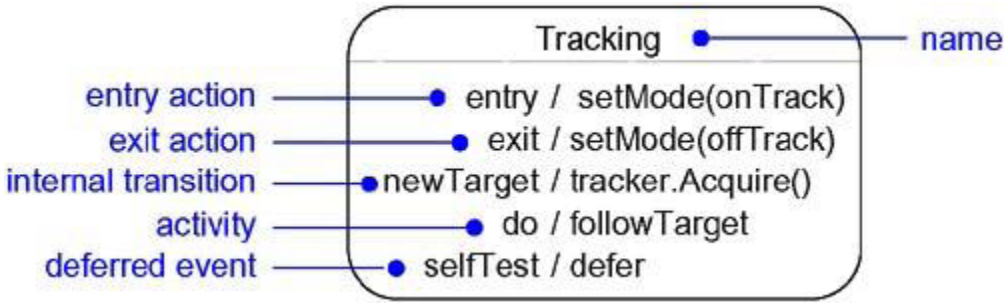


Figure 4: Advanced States and Transitions

Entry and Exit Actions

- Entry Actions are those actions that are to be done upon entry of a state and are shown by the keyword event ‘entry’ with an appropriate action
- Exit Actions are those actions that are to be done upon exit from a state marked by the keyword event ‘exit’, together with an appropriate action

Internal Transitions

- Internal Transitions are events that should be handled internally without leaving the state.
- Internal transitions may have events with parameters and guard conditions.

Activities

Activities make use of object’s idle time when inside a state. ‘do’ transition is used to specify the work that’s to be done inside a state after the entry action is dispatched.

Deferred Events

A deferred event is a list of events whose occurrence in the state is postponed until a state in which the listed events are not deferred becomes active, at which time they occur and may trigger transitions as if they had just occurred. A deferred event is specified by listing the event with the special action ‘defer’.

Substates

- A substate is a state that’s nested inside another one.
- A state that has substates is called a composite state.
- A composite state may contain either *concurrent (orthogonal) or sequential (disjoint) substates*.
- Substates may be nested to any level

Sequential Substates

- Sequential Substates are those substates in which an event common to the composite states can eaisly be excercised by each states inside it at any time
- sequential substates partition the state space of the composite state into disjoint states
- Figure 5: shows Sequential Substates
- A nested sequential state machine may have at most one initial state and one final state

History States

- A history state allows composite state that contains sequential substates to remember the last substate that was active in it prior to the transition from the composite state.
- a shallow history state is represented as a small circle containing the symbol H
- Figure 6: shows History State
- The first time entry to a composite state doesn’t have any history and the process for collecting history is as shown in the figure: 6
- the symbol H designates a *shallow history*, which remembers only the history of the immediate nested state machine.
- the symbol H* designates *deep history*, which remembers down to the innermost nested state at any depth.
- When only one level of nesting, shallow and deep history states are semantically equivalent.

Concurrent Substates

- concurrent substates specify two or more state machines that execute in parallel in the context of the enclosing object
- Figure 7: shows Concurrent Substates
- Execution of these concurrent substates continues in parallel. These substates waits for each other to finish to joins back into one flow
- A nested concurrent state machine does not have an initial, final, or history state

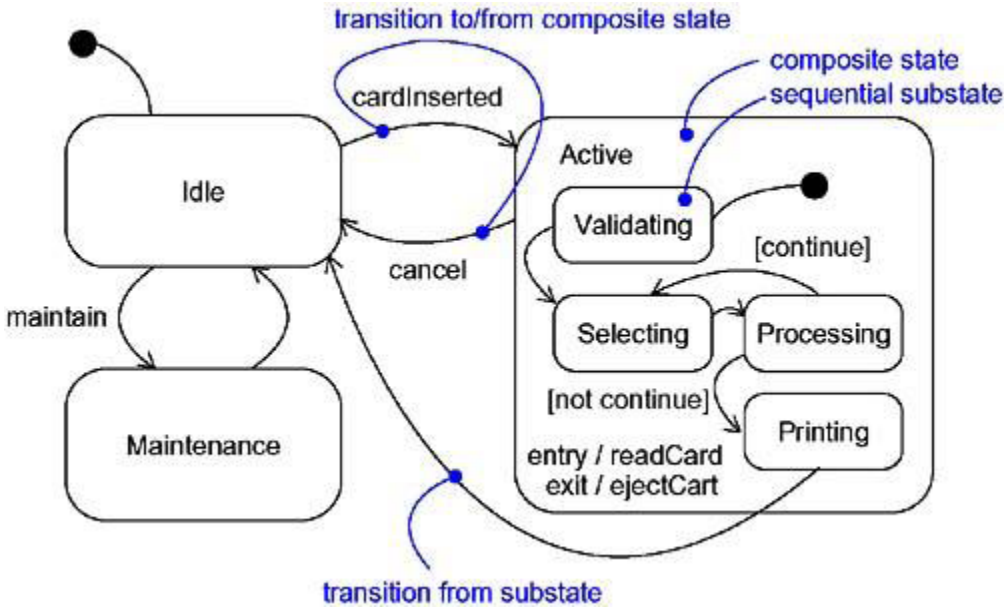


Figure 5: Sequential Substates

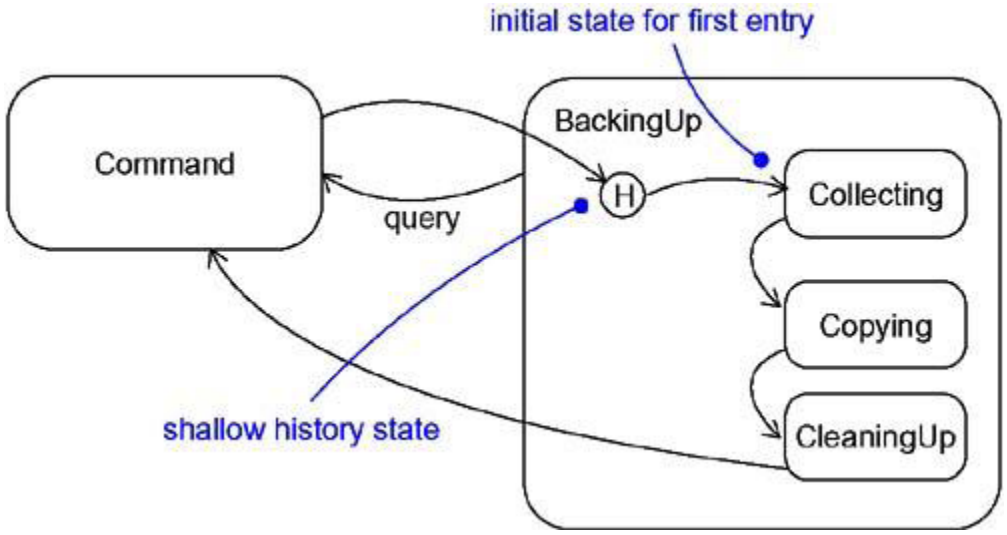


Figure 6: History State

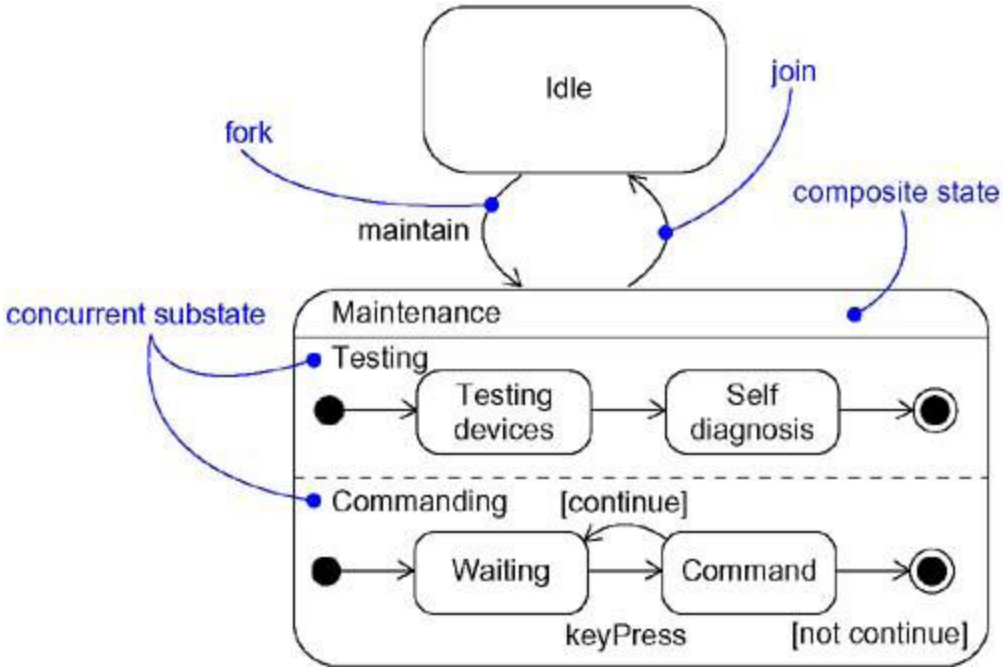


Figure 7: Concurrent Substates

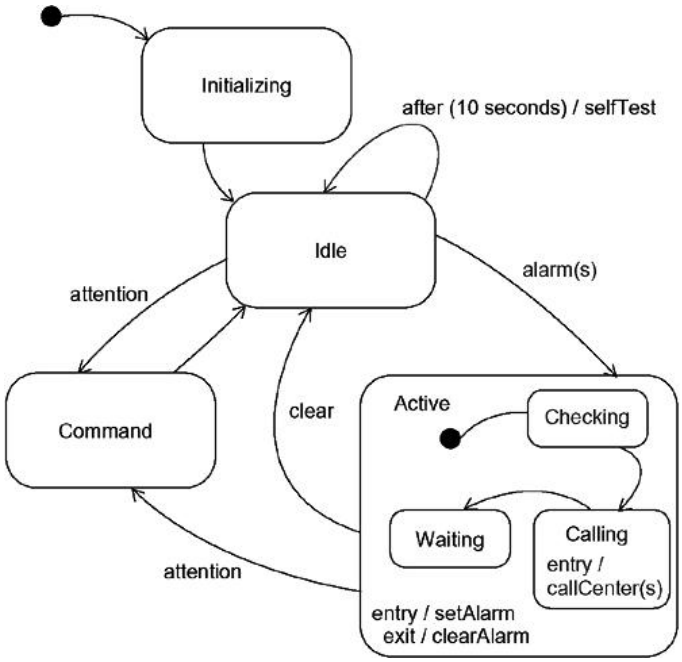
Modeling the Lifetime of an Object

To model the lifetime of an object,

- · Set the context for the state machine, whether it is a class, a use case, or the system as a whole.
- *If the context is a class or a use case*, collect the neighboring classes, including any parents of the class and any classes reachable by associations or dependences. These neighbors are candidate targets for actions and are candidates for including in guard conditions.
- *If the context is the system as a whole*, narrow your focus to one behavior of the system. Theoretically, every object in the system may be a participant in a model of the system's lifetime, and except for the most trivial systems, a complete model would be intractable.
- · Establish the initial and final states for the object. To guide the rest of your model, possibly state the pre- and postconditions of the initial and final states, respectively.
- · Decide on the events to which this object may respond. If already specified, you'll find these in the object's interfaces; if not already specified, you'll have to consider which

objects may interact with the object in your context, and then which events they may possibly dispatch.

- · Starting from the initial state to the final state, lay out the top-level states the object may be in. Connect these states with transitions triggered by the appropriate events. Continue by adding actions to these transitions.
- · Identify any entry or exit actions (especially if you find that the idiom they cover is used in the state machine).
- · Expand these states as necessary by using substates.
- · Check that all events mentioned in the state machine match events expected by the interface of the object. Similarly, check that all events expected by the interface of the object are handled by the state machine. Finally, look to places where you explicitly want to ignore events.
- · Check that all actions mentioned in the state machine are sustained by the relationships, methods, and operations of the enclosing object.
- · Trace through the state machine, either manually or by using tools, to check it against expected sequences of events and their responses. Be especially diligent in looking for unreachable states and states in which the machine may get stuck.
- · After rearranging your state machine, check it against expected sequences again to ensure that you have not changed the object’s semantics.
  For example, Figure 8 shows the state machine for the controller in a home security system, which is responsible for monitoring various sensors around the perimeter of the house



Source : http://praveenthomasln.wordpress.com/2012/04/07/state-machines-s8-cs/