WHITESTEIN
Technologies

Unified Modeling Language

State Machines

Radovan Cervenka

State Machine Model

→ **Used for modeling discrete behavior through finite state transition systems (called also final state machine - FSM).**

- ***Behavioral state machines***—used to specify behavior (life cycle) of various model elements, e.g., classes, instances, subsystems, or components.
- ***Protocol state machines***—used to express *usage protocols*, i.e., legal usage scenarios of classifiers, interfaces, or ports.
- The UML's state machine formalism is an object-based variant of *Harel statecharts*.

Consists of:

- State machine diagrams.
- Element descriptions.

Used (mainly) in:

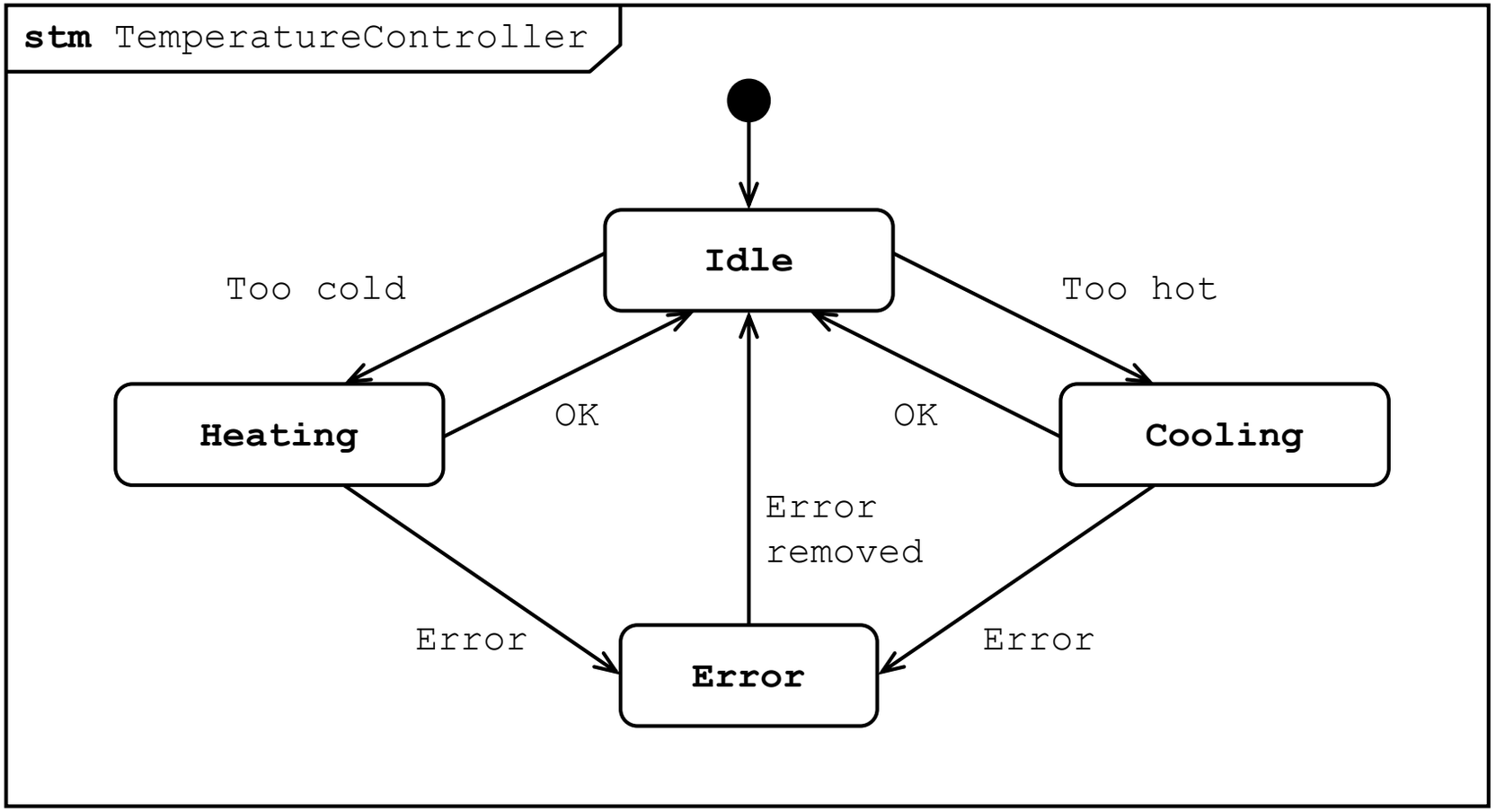
- Analysis and design ⇒ specification of behavior of various model elements, or specification of usage protocols.

State Machine

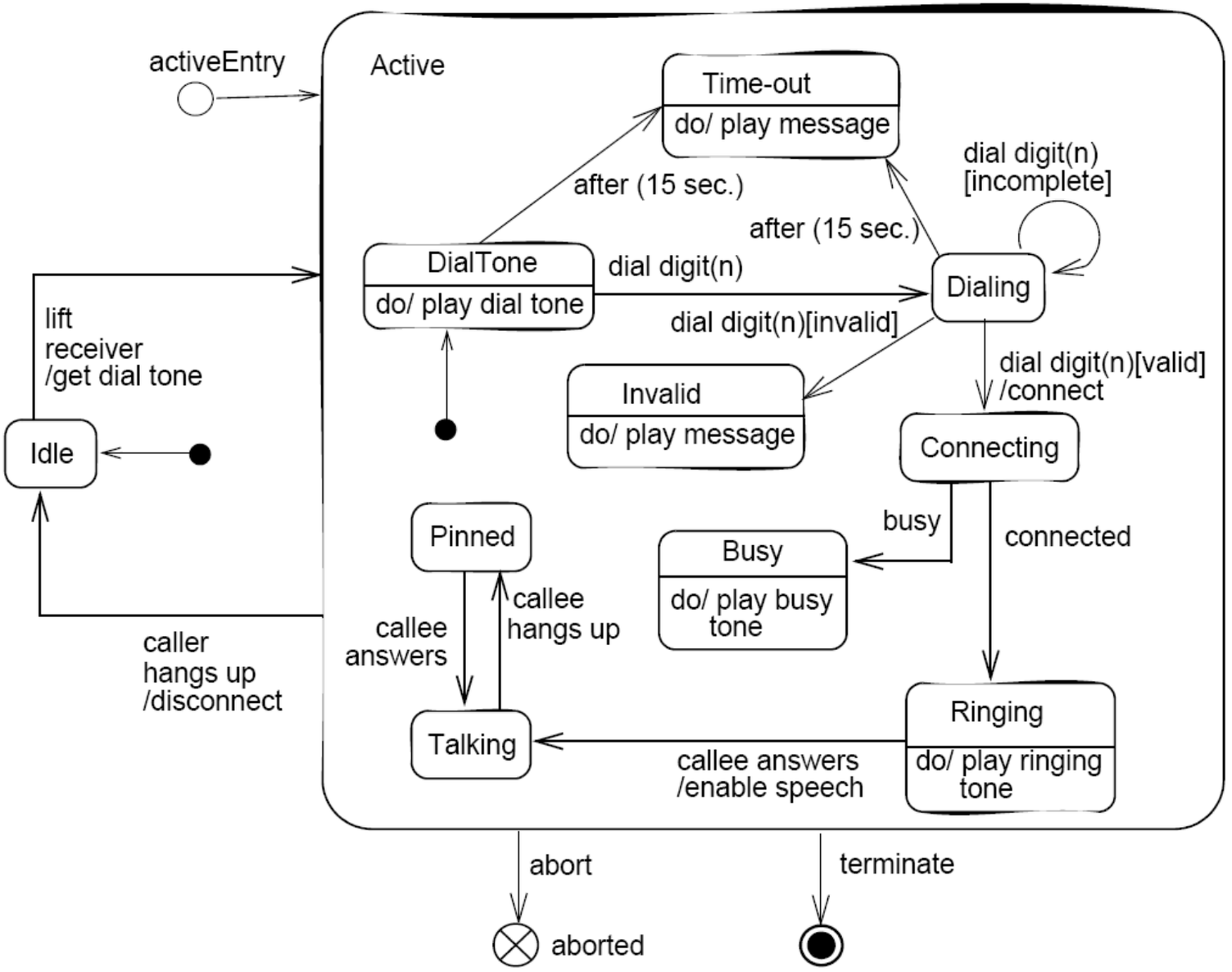
→ Specifies the behavior of part of a system in the form of final state automata.

- Behavior is modeled as a traversal of a graph of state nodes interconnected by one or more joined transition arcs that are triggered by the dispatching of series of (event) occurrences.
- During this traversal, the state machine executes a series of activities associated with various elements of the state machine.
- A state machine owns one or more regions, which in turn own vertices and transitions. Each region determines an independent state machine which runs in parallel to the state machines in other regions.
- The *behaviored classifier* owning a state machine defines which signal and call triggers are defined for the state machine, and which attributes and operations are available in activities of the state machine.
  - A state machine without a context classifier may use triggers that are independent of receptions or operations of a classifier.
- A state machine is drawn as a state machine diagram with its contained nodes and vertices. The content can optionally be placed within a diagram frame having the label "***stm***" before name.

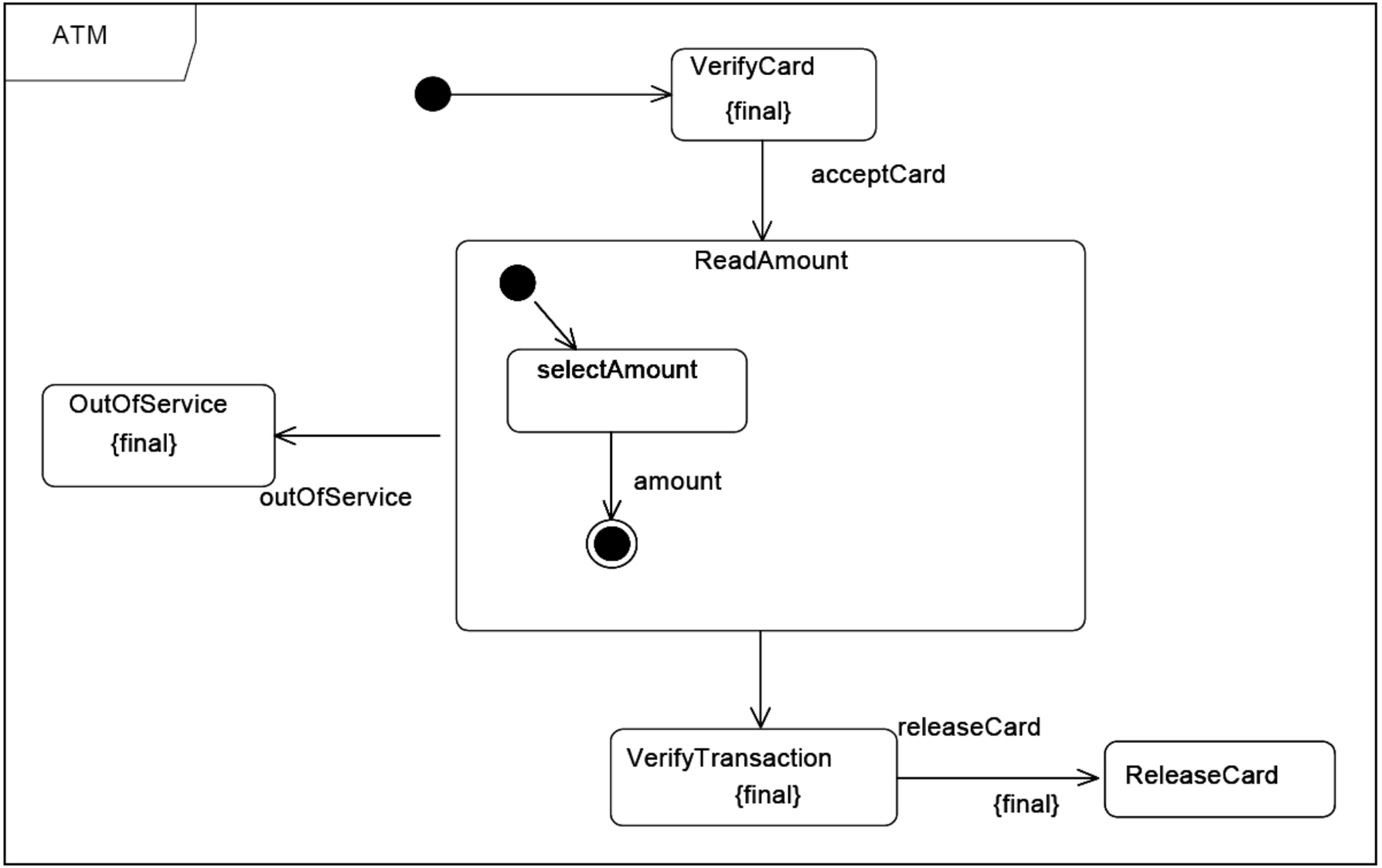
Examples of State Machines (1)



Examples of State Machines (2)



Examples of State Machines (3)



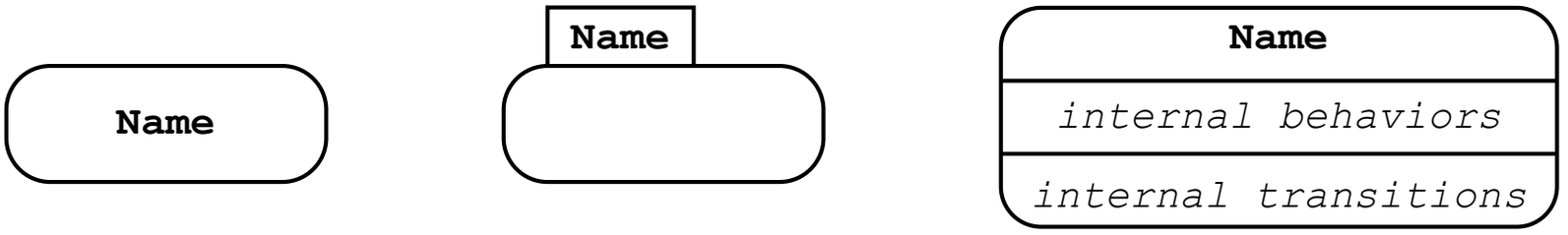
(Simple) State

→ Represents a situation during which some (usually implicit) invariant condition holds.

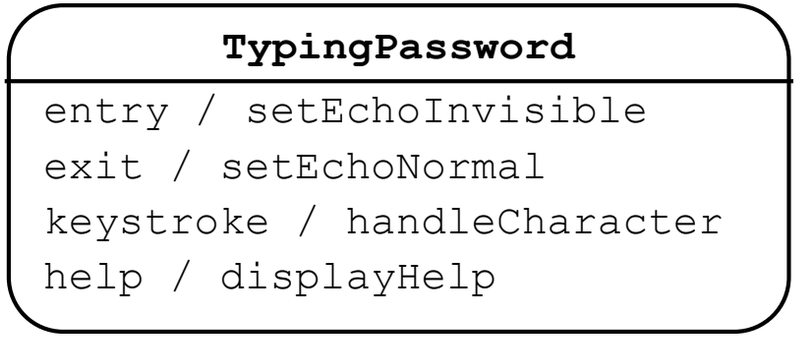
- The invariant may represent a static situation (such as an object waiting for some external event to occur) or a dynamic condition (such as the process of performing some behavior).
- A state becomes active when it is entered as a result of some transition, and becomes inactive if it is exited as a result of a transition.
- ***Internal behaviors***–executed within the state.
  - Format: *event* '/' *behavior-expression*
  - Special events:
    - *entry*: entry to the state; specifies the *entry behavior*.
    - *exit*: exit from the state; specifies the *exit behavior*.
    - *do*: as long as the modeled element is in the state or until the computation specified by the expression is complete; specifies the *do activity* (behavior).

(Simple) State (cont.)

- ***Internal transitions***–executed without exiting or re-entering the state in which they are defined.
  - Format as for transitions.
- ***Deferrable events***–do not trigger any transitions in the current state, but remain in the event pool ready for processing by another state or transition.
  - Format: *event* '/' 'defer'.



Example of Simple State



Composite State

→ A composite state either:

- contains one region (called ***non-orthogonal composite state***), or
- is decomposed into two or more orthogonal regions (called ***orthogonal composite state***).

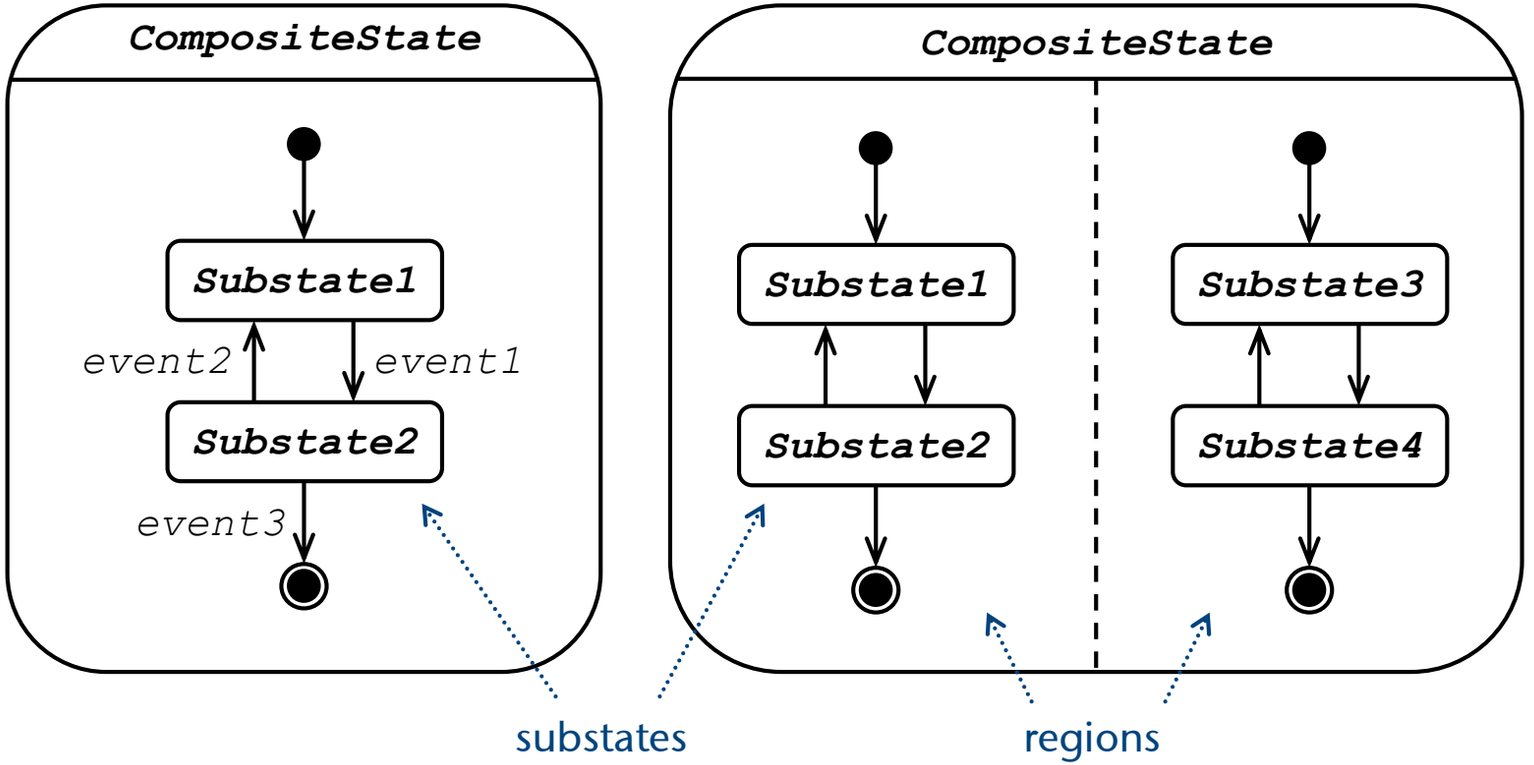
- Each region can contain a set of mutually exclusive disjoint *substates* and a set of transitions.
- If the composite is active, each of its owned regions can have at most one immediately contained substate active. Therefore, if a state is active, all its transitively contained states are also active.
- A transition to the enclosing state represents a transition to the initial pseudostate in each region.
- A transition going directly to a substate activates it explicitly and all other orthogonal regions (if any) are activated in their initial pseudostates.
- A transition to a final state of the region represents the completion of behavior in the enclosing region.
- Completion of behavior in all orthogonal regions represents completion of behavior by the enclosing composite state.

Composite State (cont.)

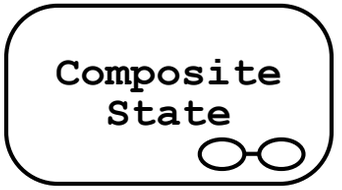
- When exiting from a composite state, the active substates are exited recursively.
  - The exit behaviors are executed in sequence starting with the innermost active states in the current state configuration.

Non-orthogonal composite sate:

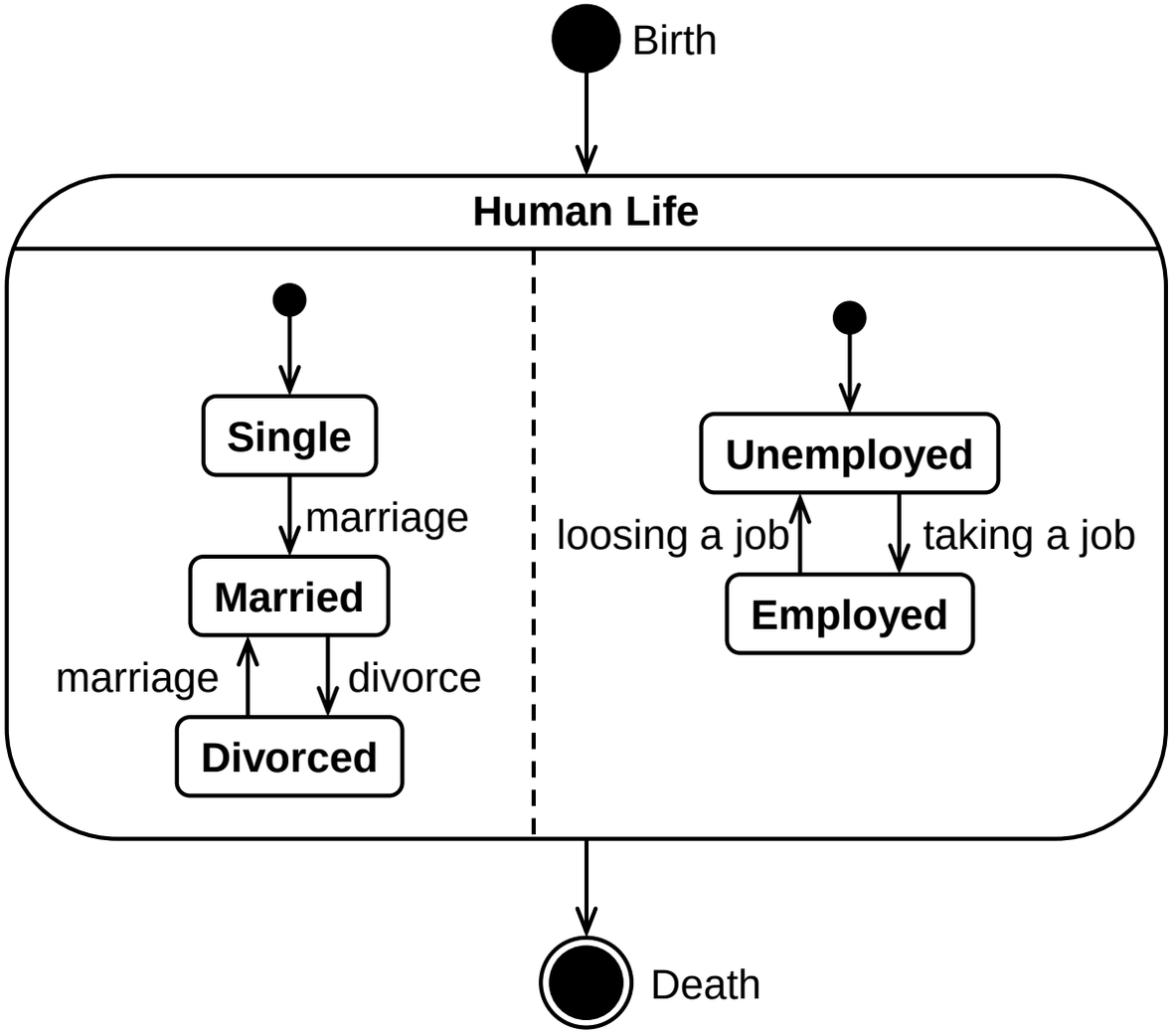
Orthogonal composite sate:



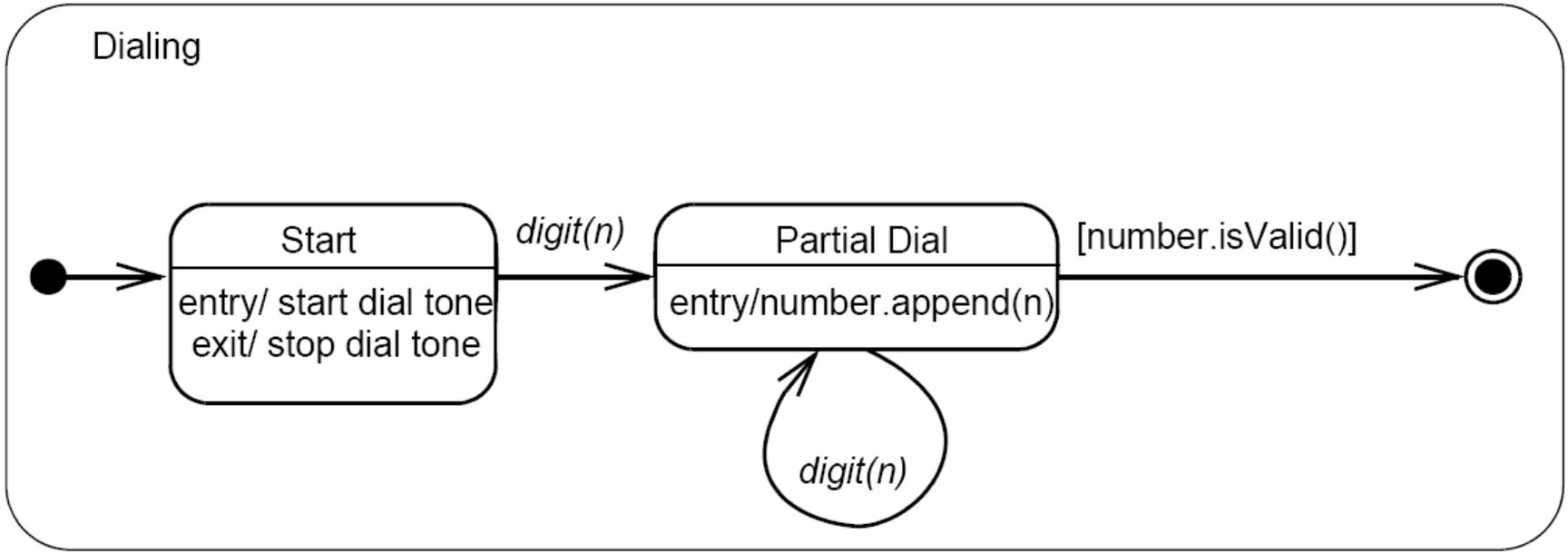
Composite sate with hidden decomposition:

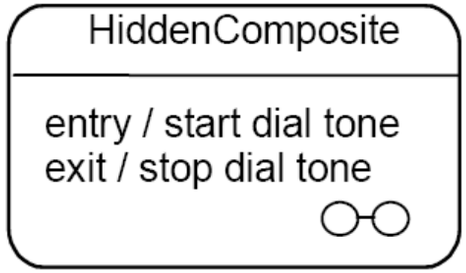


Examples of Composite States (1)

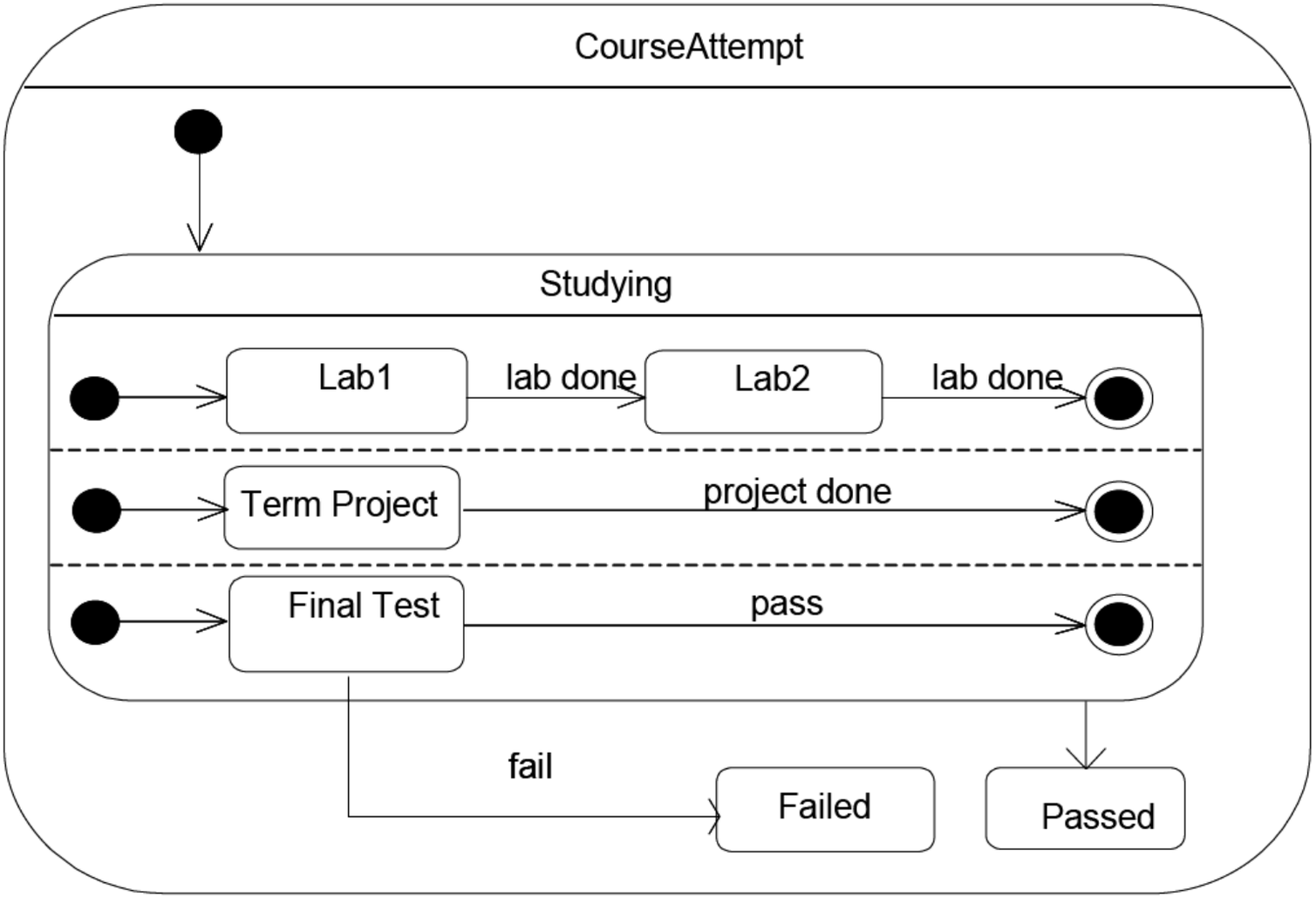


Examples of Composite States (2)





Examples of Composite States (3)

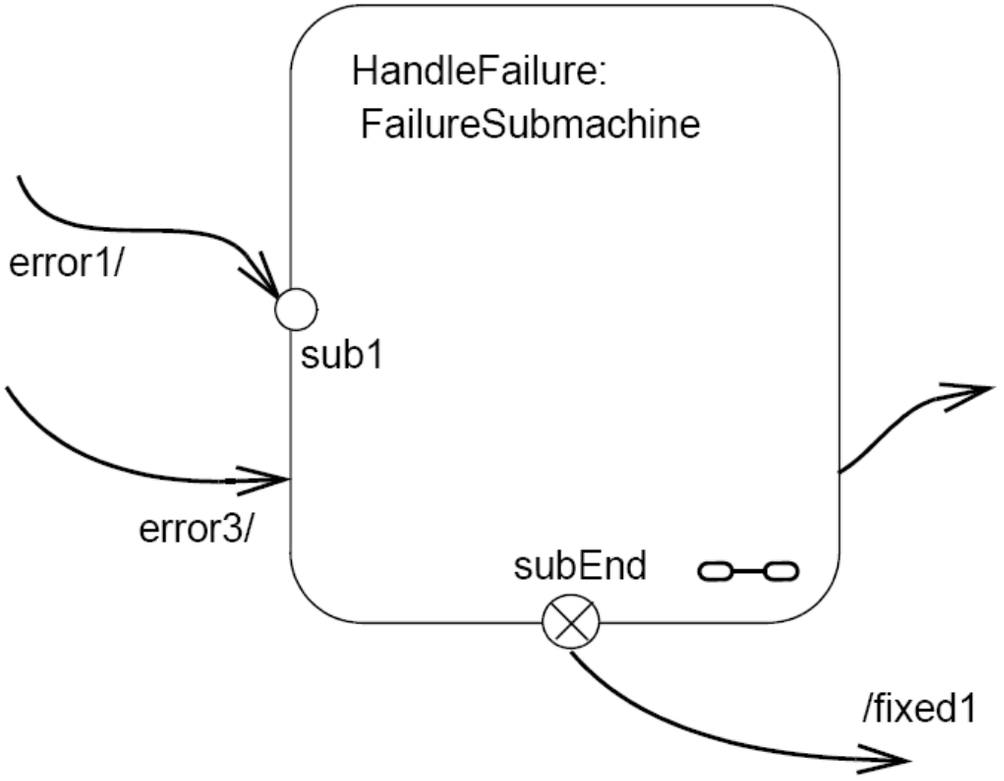


Submachine State

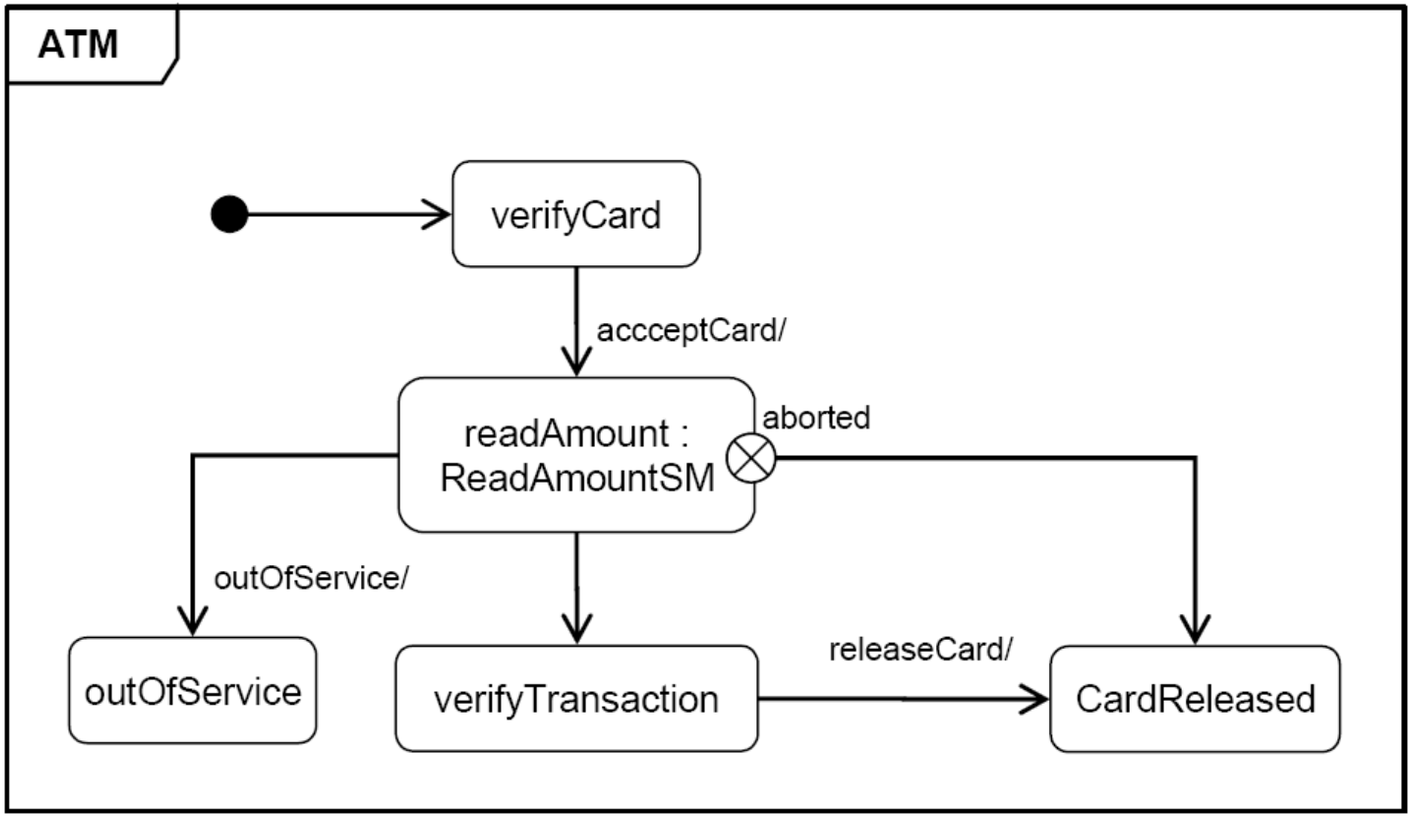
→ Specifies the insertion of the specification of a submachine state machine.

- Semantically equivalent to a composite state.
  - Regions of the submachine state machine are the regions of the composite state.
  - Transitions in the containing state machine can have entry/exit points of the inserted state machine as targets/sources.
- The same state machine may be a submachine more than once in the context of a single containing state machine.

Examples of Submachine States (1)

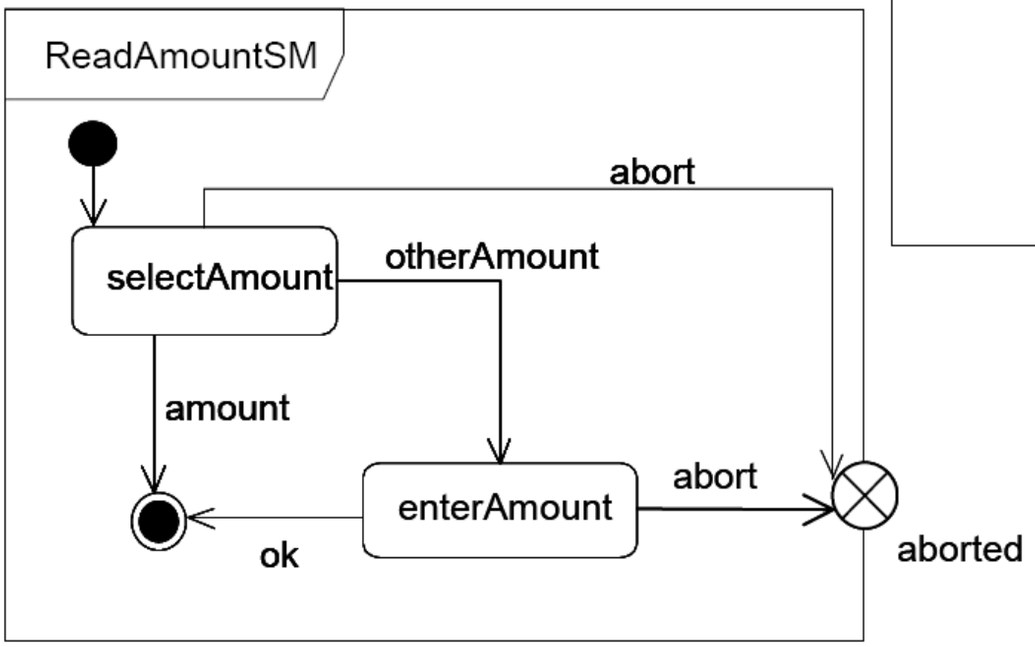


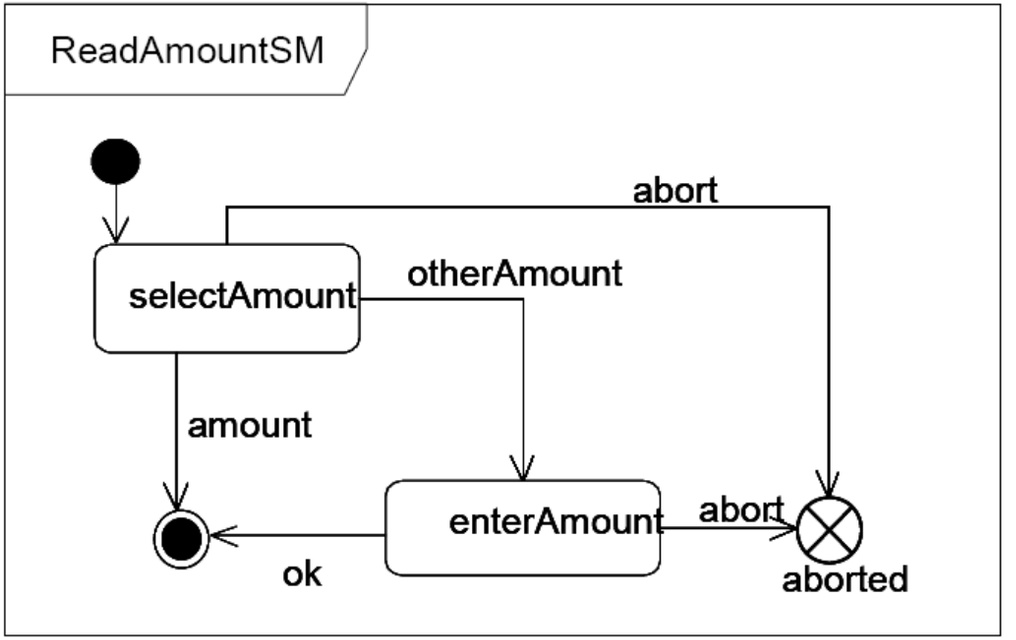
Examples of Submachine States (2)



Examples of Submachine States (3)

Two alternatives of state machine referable from a submachine state:



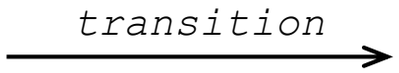


Transition

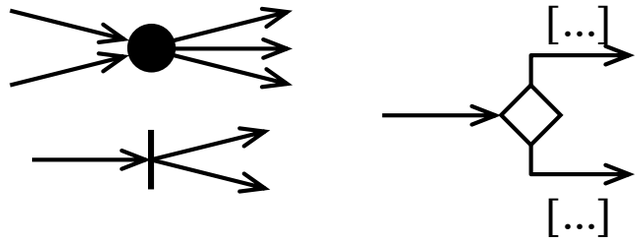
→ A directed relationship between a source vertex and a target vertex, which takes the state machine from one state configuration to another, representing the response to an occurrence of an event of a particular type.

- ***High-level*** (***group***) ***transition***–originating from composite states.
  - If triggered, they result in the innermost exiting of all the substates.
- ***Compound transition***–a "semantically complete" an acyclical unbroken chain of transitions joined via join, junction, choice, or fork pseudostates (see later) that define path from a set of source states to a set of destination states.
  - A ***simple transition*** connecting two states is therefore a special common case of a compound transition.
- ***Completion transition***–a transition originating from a state or an exit point which does not have an explicit trigger and is implicitly triggered by a completion of its source.

Simple transition:



Compound transitions:



Transition (cont.)

- Format:

  transition ::= [*trigger* [‘,’ *trigger*]* [‘[‘ *guard* ‘]’] [‘/’ *behavior-expression*]]

  - *trigger* specifies the event which fires the transition

    trigger ::= *name* [‘(‘ [*attr-spec* [‘,’ *attr-spec*]*] ‘)’] |
    ‘all’ | (‘after’ | ‘at’ | ‘when’) *expression*

    attr-spec ::= *attr-name* [‘:’ *type-name*]

    – *attr-spec* specifies the event attribute; its name and type

    – ‘all’ determines the accepting of all events

    – ‘after’, ‘at’ and ‘when’ determine the time event relative to the specified time

  - *guard* specifies a condition which enables firing the transition

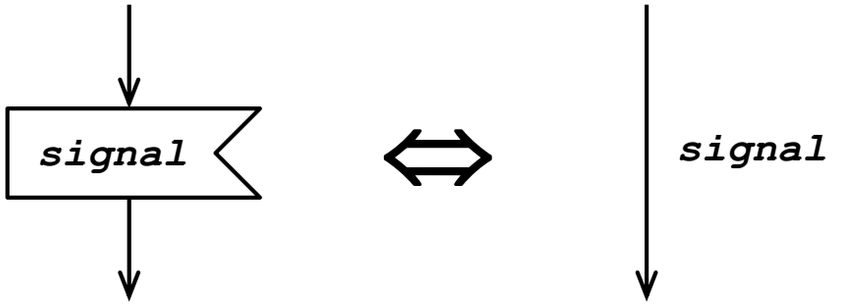
  - *behavior-expression* specifies the behavior which is executed by firing the transition

- Example:

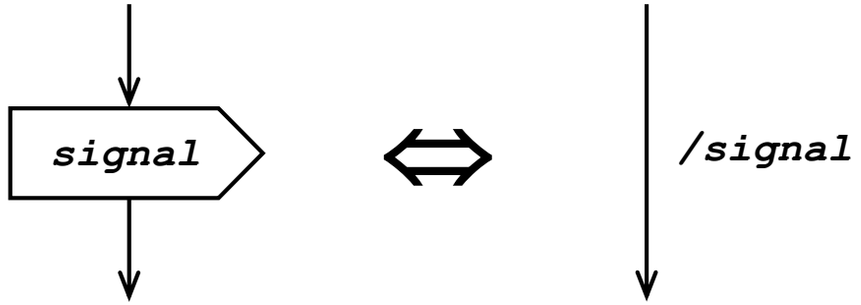
```
right-mouse-down(location) [location in window] /
   object := pick-object(location);object.highlight()
```

Transition–Presentation Options

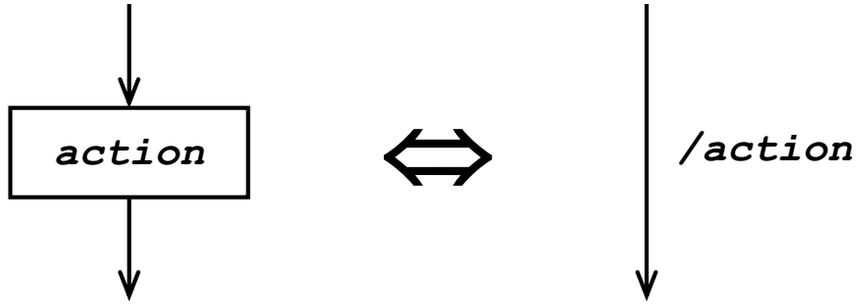
- Trigger = signal receipt



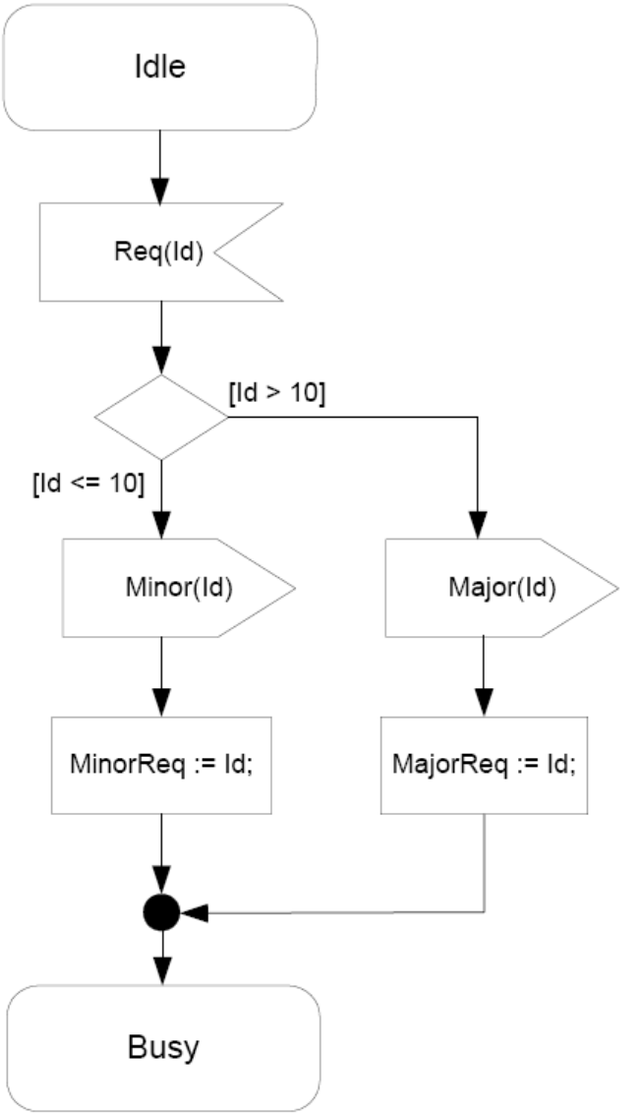
- Behavior = send signal action



- Behavior = other action



Examples of Transitions



Initial

- → A pseudostate representing a default vertex that is the source for a single transition to the default state of a region.
- There can be at most one initial vertex in a region.
- Can have at most one outgoing transition.
- The outgoing transition from the initial vertex may have a behavior, but not a trigger or guard.

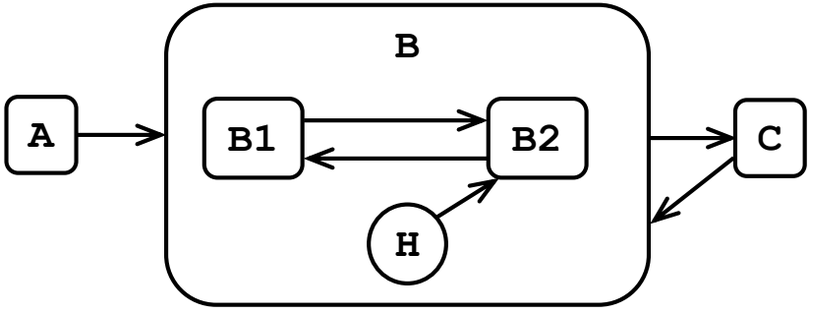
Final State

→ A special kind of state signifying that the enclosing region is completed.

- If the enclosing region is directly contained in a state machine and all other regions in the state machine also are completed, the entire state machine is completed.

Shallow History

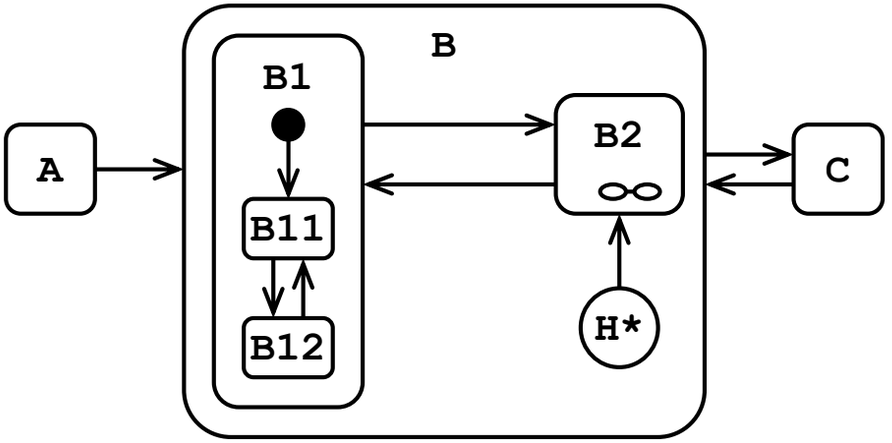
- → A pseudostate representing the most recent active substate of its containing state, but not the substates of that substate.
- A composite state can have at most one shallow history vertex.
- A transition coming into the shallow history vertex is equivalent to a transition coming into the most recent active substate of a state.
- At most one transition may originate from the history connector to the *default shallow history state*. This transition is taken in case the composite state had never been active before.



Deep History

→ A pseudostate representing the most recent active configuration (including recursive substates) of the composite state that directly contains this pseudostate, i.e., the state configuration that was active when the composite state was last exited.

- A composite state can have at most one deep history vertex.
- At most one transition may originate from the history connector to the *default deep history state*. This transition is taken in case the composite state had never been active before.



Entry and Exit Points

Entry Point

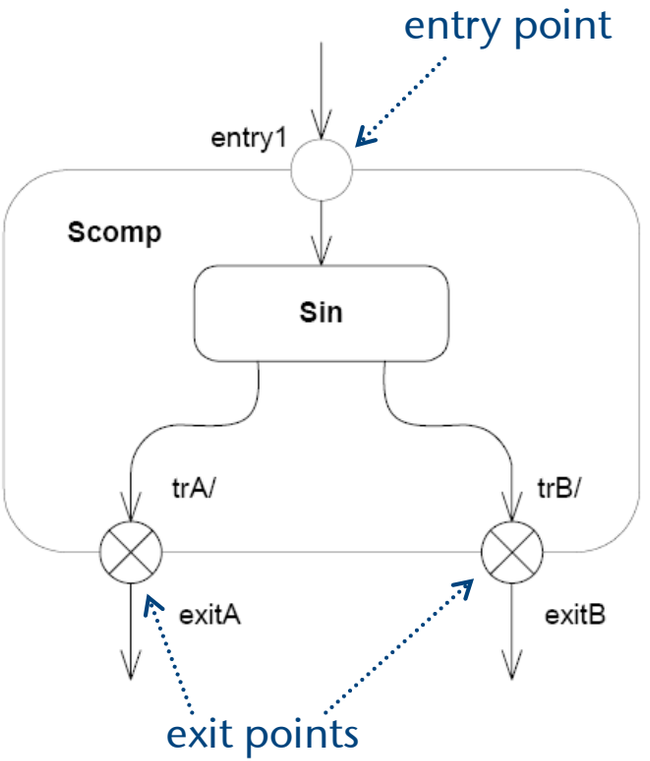
→ A pseudostate which is an entry point of a state machine or a composite state.

- In each region of the state machine or composite state it has a single transition to a vertex within the same region.

Exit Point

→ A pseudostate which is an exit point of a state machine or composite state.

- Entering an exit point within any region of the composite state or state machine referenced by a submachine state implies the exit of this composite state or submachine state and the triggering of the transition that has this exit point as source in the state machine enclosing the submachine or composite state.



Fork and Join

Fork

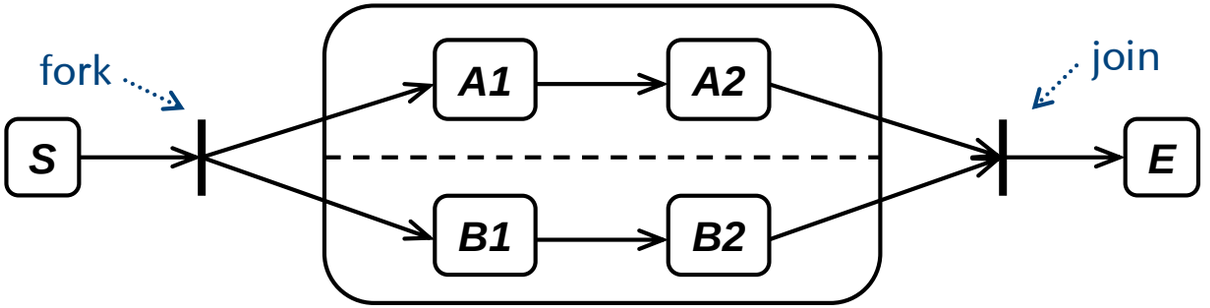
→ A pseudostate used to split an incoming transition into two or more transitions terminating on orthogonal target vertices (i.e., vertices in different regions of a composite state).

- The segments outgoing from a fork vertex must not have guards or triggers.
- Must have exactly one incoming and at least two outgoing transitions.

Join

→ A pseudostate used to merge several transitions emanating from source vertices in different orthogonal regions.

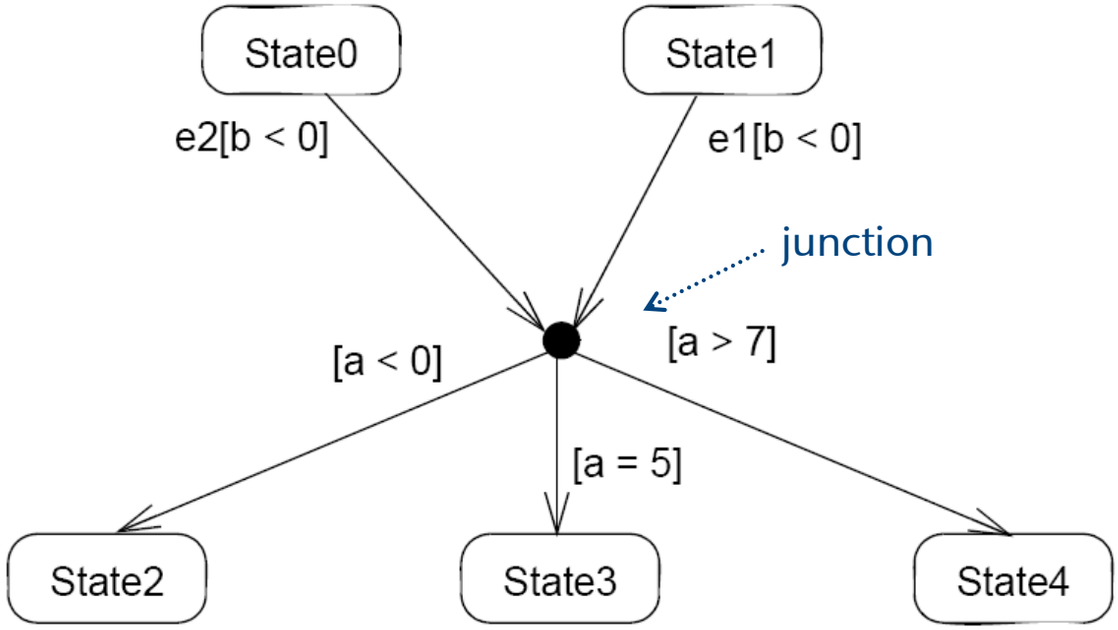
- The transitions entering a join vertex cannot have guards or triggers.
- Must have at least two incoming transitions and exactly one outgoing transition.



Junction

→ A pseudostate used to chain together multiple transitions.

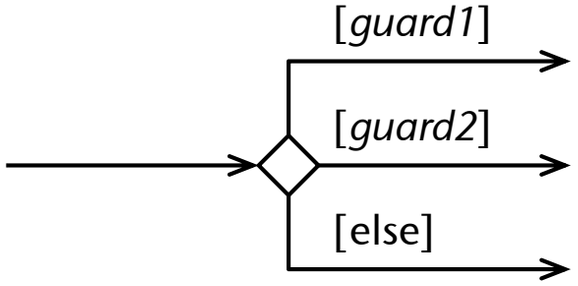
- No semantics impact, can be replaced by multiple transitions.
- Constructs compound transition paths between states used to merge and/or split transitions.
- Must have at least one incoming and one outgoing transition.



Choice

→ A pseudostate which, when reached, result in the dynamic evaluation of the guards of the triggers of its outgoing transitions. This realizes a *dynamic conditional branch*.

- Allows splitting of transitions into multiple outgoing paths such that the decision on which path to take.
- The decision may be a function of the results of prior actions performed in the same execution step.
- If more than one of the guards evaluates to true, an arbitrary one is selected.
- If none of the guards evaluates to true, the model is considered ill-formed.
- Predefined “else” guard can be used for one outgoing transition.
- Must have at least one incoming and one outgoing transition.



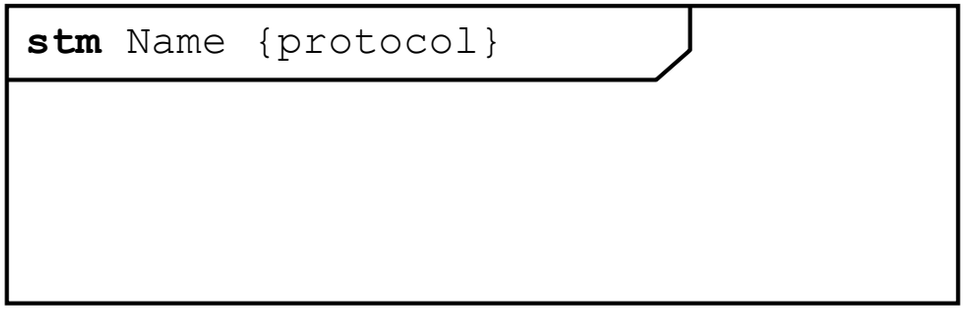
Terminate

- → A pseudostate used to terminate the execution of its state machine by means of destroying its context object.
- ■ The state machine does not exit any states nor does it perform any exit actions other than those associated with the transition leading to the terminate pseudostate.

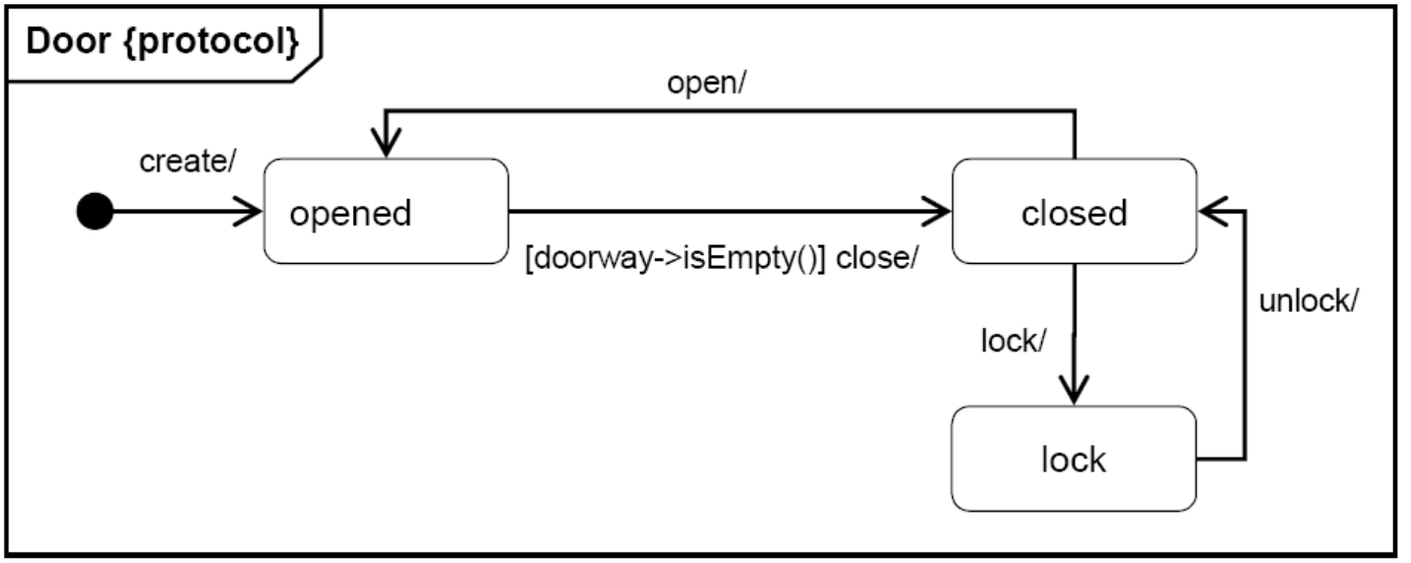
Protocol State Machine

→ A variant of state machine that specifies which operations of the *context classifier* can be called in which state and under which condition, i.e., determines the allowed call sequences on the classifier's operations.

- Presents the possible and permitted transitions on the instances of its context classifier, together with the operations that carry the transitions.
  - In this way an instance usage *lifecycle* can be created.
- All transitions of a protocol state machine must be protocol transitions.
- Protocol state machines cannot have deep or shallow history pseudostates.



Example of Protocol State Machine



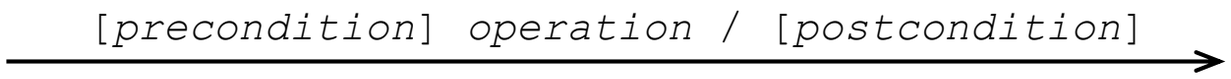
State in Protocol State Machines

- An exposed stable situation of its context classifier.
- The state represents the situation when an instance of the classifier is not processing any operation.
- Can specify an *invariant*–condition(s) that is(are) always true when an instance of the classifier is in the current state.
  - State invariants are additional conditions to the preconditions of the outgoing transitions, and to the postcondition of the incoming transitions.
- Cannot have entry, exit, or do internal behaviors.

Protocol Transition

→ A specialized transition that specifies a legal transition for an operation.

- Specifies:
  - ***Pre-condition***: the condition that must be true at triggering the transition.
  - ***Operation***: a reference to the context's operation which represents its call at the given position in the protocol state machine.
  - ***Post-condition***: the condition that should be obtained once the transition is accomplished.
- No behavior (action) is specified for protocol transition.



Overall Semantics of State Machines

- Event occurrences go into the (FIFO) *event pool*.
- Event occurrences are processed one at a time, but only if processing of the previous event completed; except of *do activities* of states (see later). This is called ***run-to-completion*** mechanism.
- One event occurrence may result in one or more transitions being enabled for firing (each in one orthogonal region).
  - All enabled (non-conflicting) transitions are fired.
  - The order in which selected transitions fire is not defined.
  - When all orthogonal regions have finished executing the transition, the current event occurrence is fully consumed.
- If no transition is enabled, the event occurrence is discarded.
- During a transition, a number of actions may be executed. If such an action is a synchronous operation invocation on an object executing a state machine, then the transition step is not completed until the invoked object completes its action.

Overall Semantics of State Machines (cont.)

- In case of conflicting transitions (for instance, two transitions with the same event and the guards both satisfied, originating from the same state), only one of them will fire.
  If *t1* is a transition whose source state is *s1*, and *t2* has source *s2*, then:
  - If *s1* is a direct or transitively nested substate of *s2*, then *t1* has higher priority than *t2* $\Rightarrow$ *t1* is fired.
  - If *s1* and *s2* are not in the same *state configuration* (an execution state of the state machine), then there is no priority difference between *t1* and *t2*.

Process of State Machine Modeling

1. Identify possible states.
2. Identify external events which can cause changes of states.
3. To each state attach transitions based on relevant events.
4. Add internal actions and transitions.
5. Identify composite states and specify contents of regions.
6. Repeat until the state machine is complete.
7. Review the state machine by simulation of the owner's lifecycle. Based on simulated events, check the appropriate triggering of transitions and execution of behavior.