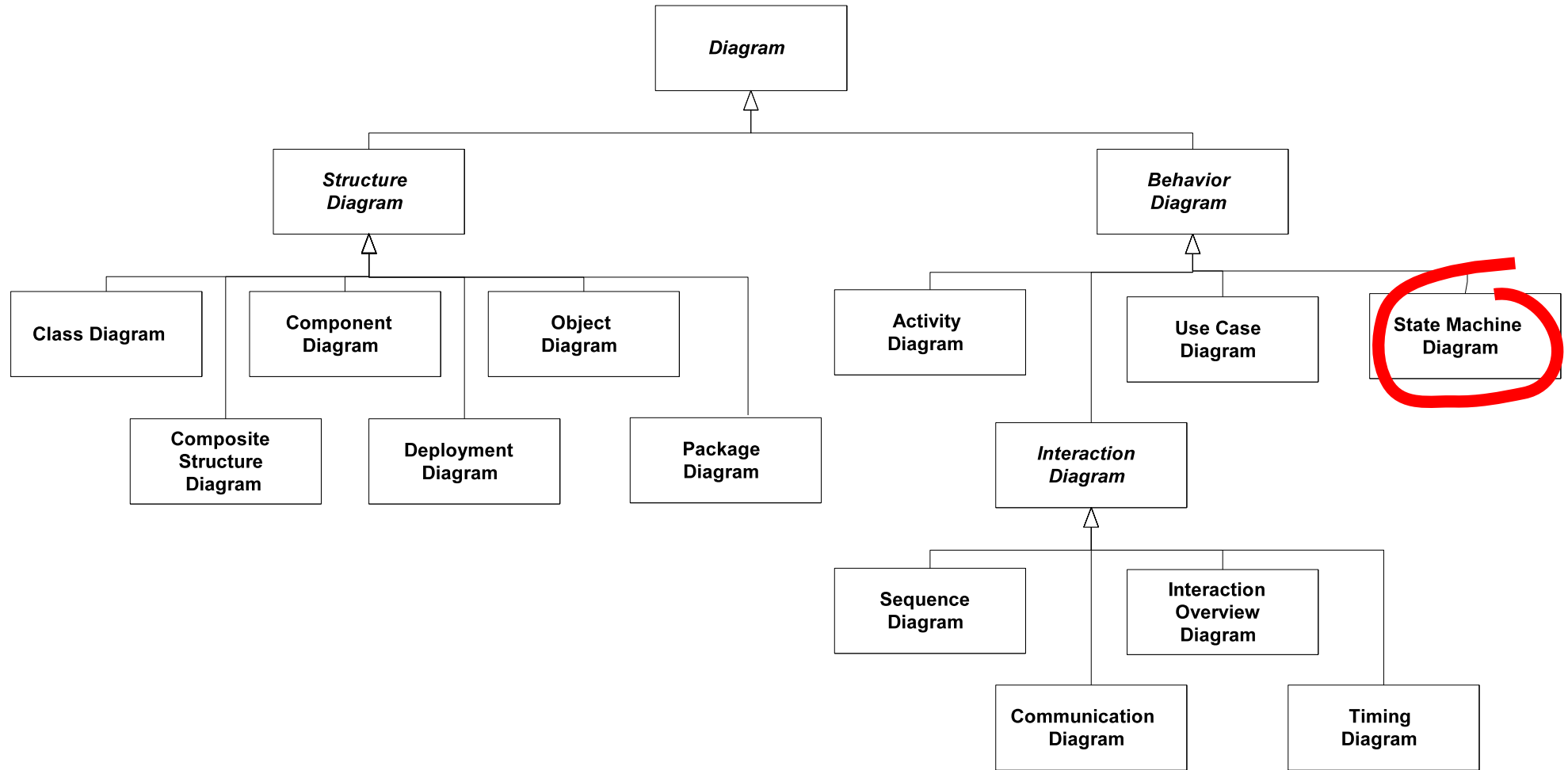


# UML: State machine diagram

Davide Rossi

Dipartimento di Informatica – Scienze e Ingegneria  
Università di Bologna



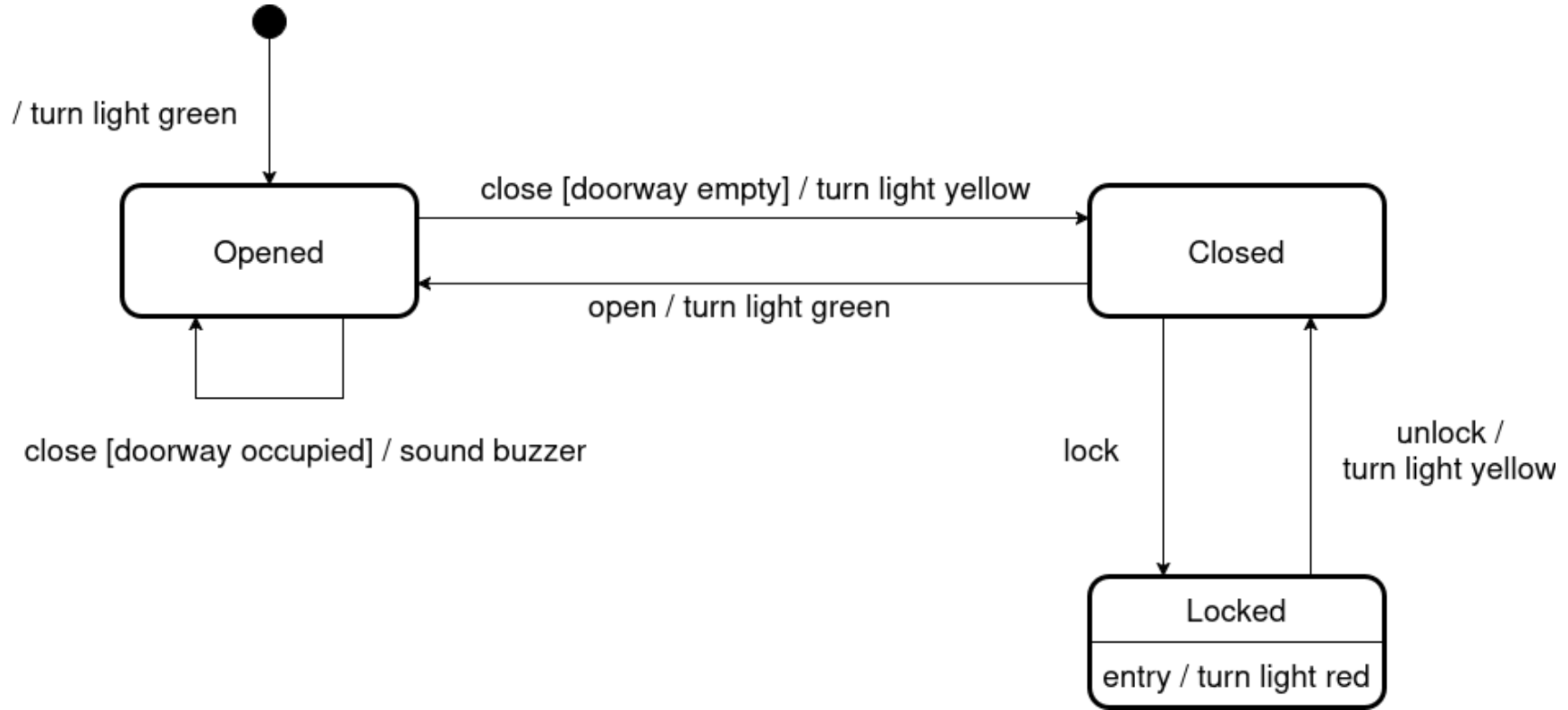


# State machine

A **behavioral state machine** describes a discrete event-driven behavior of a system or a part of a system as the traversal of a graph of vertices (usually states) connected by transitions.

A **protocol state machine** describes the lifecycle or the valid interactions sequences (protocols) for parts of a system (a classifier).

# Example: door – behavioral SM



# State

- State: a situation in which some invariant condition holds
- Same stimulus → same response
- Same active behavior
- States can be:
  - Simple: no internal vertices or transitions
  - Composite: contains one or more regions – states in these regions are called substates
  - Submachine

# State

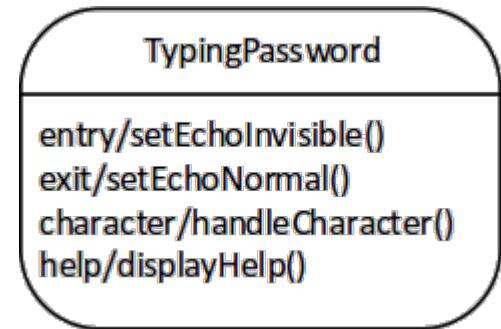
- States can be associated to the following behaviors:
  - entry
  - exit
  - doActivity
- doActivity behavior execute concurrently with any other state-associated behavior and is aborted if not finished when the state is exited

# States: notation

A State is depicted as a rounded rectangle, it may be subdivided into multiple compartments separated with a horizontal line. The compartments are:

- name compartment
- internal Behaviors compartment
- internal Transitions compartment
- decomposition compartment (for composite states)

Notations:

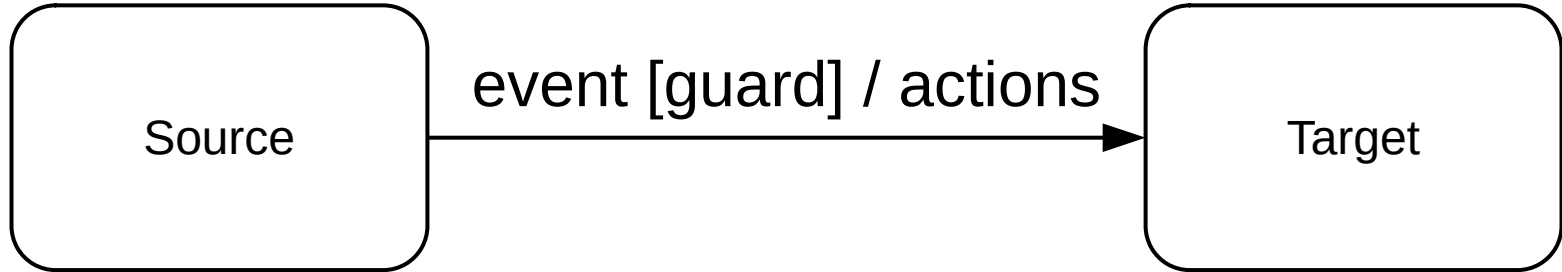


# Transition

- Transition: describes the (atomic) passage from one state to another
- Transitions are **triggered by events**. During the traversal the state machine can execute some activities
- Transitions can have guards (conditions), if the guard is false the event is discarded and the transition does not take place
- Transition with guards but no events are evaluated when the internal behavior of the source state is terminated (that is, the event is the completion of the internal behavior)



# Transition: notation



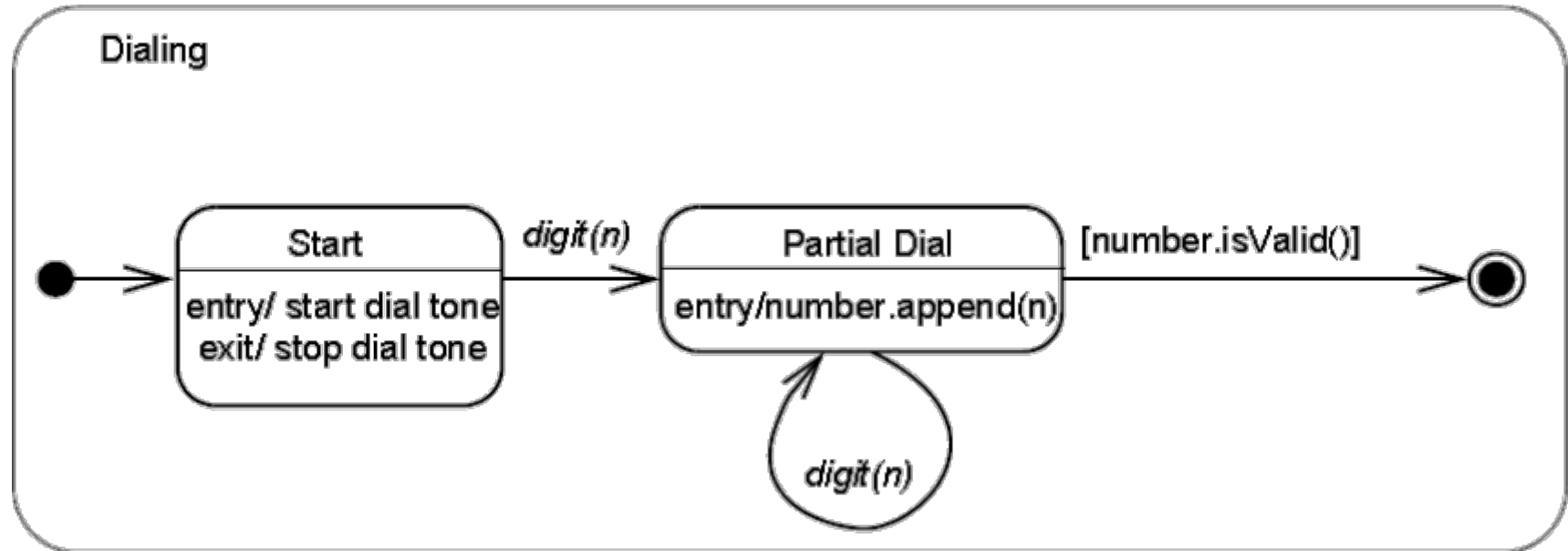
# Event

- Event: observable occurrence (in the environment of the subject)
- Takes place at a point in time (has no duration)
- May have parameters
- Basic event types are:
  - MessageEvent (CallEvent, SignalEvent)
  - ChangeEvent
  - TimeEvent

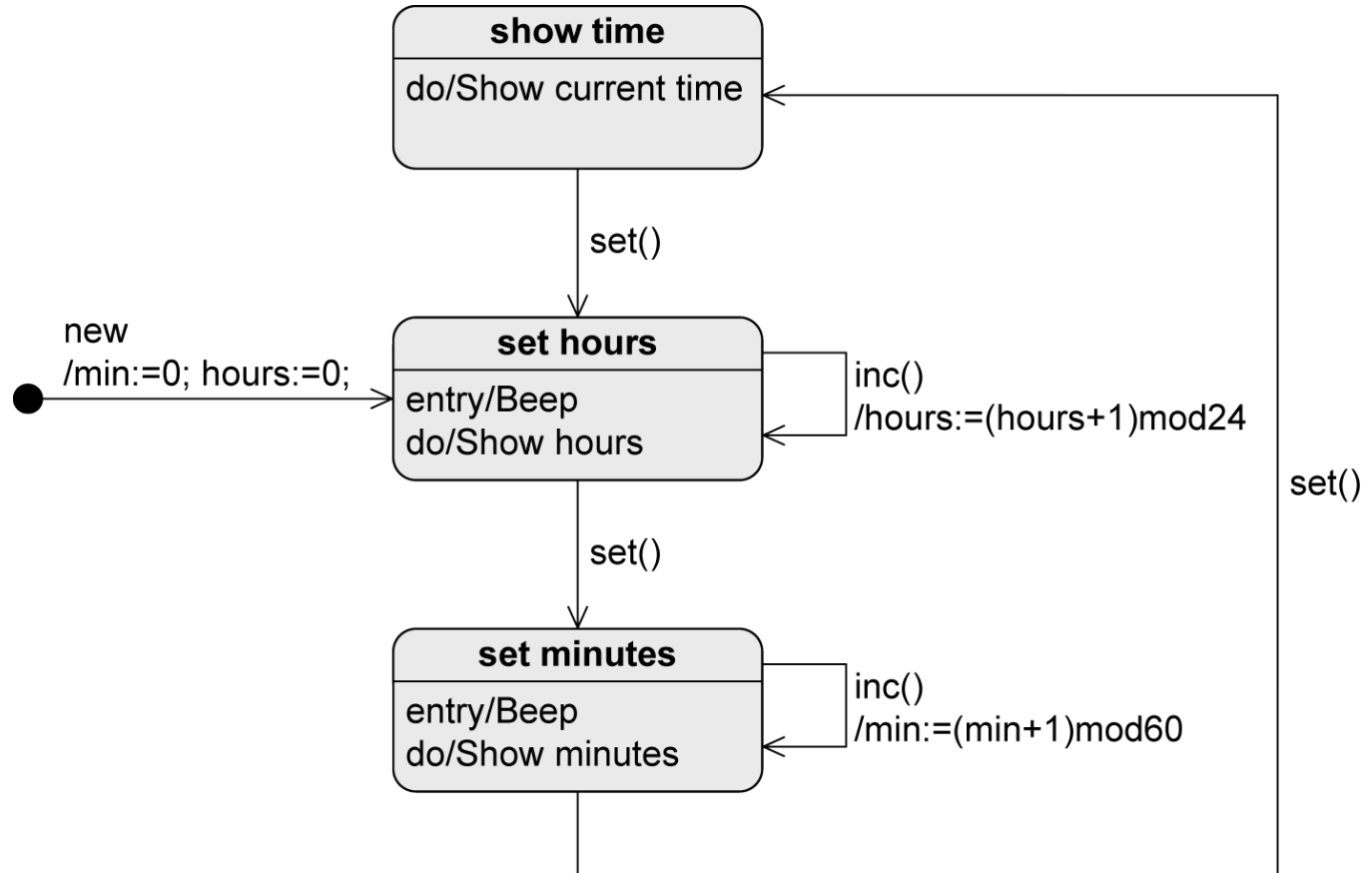
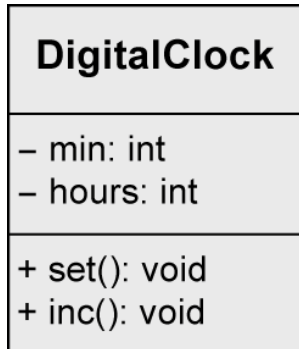
# Final state and pseudostates

- The final state is a special kind of state signifying that the enclosing region has completed.
- Several pseudostates are used to enrich the semantics of the state machine:
  - initial
  - join
  - fork
  - junction
  - choice
  - entryPoint
  - exitPoint
  - terminate

# Dialing example

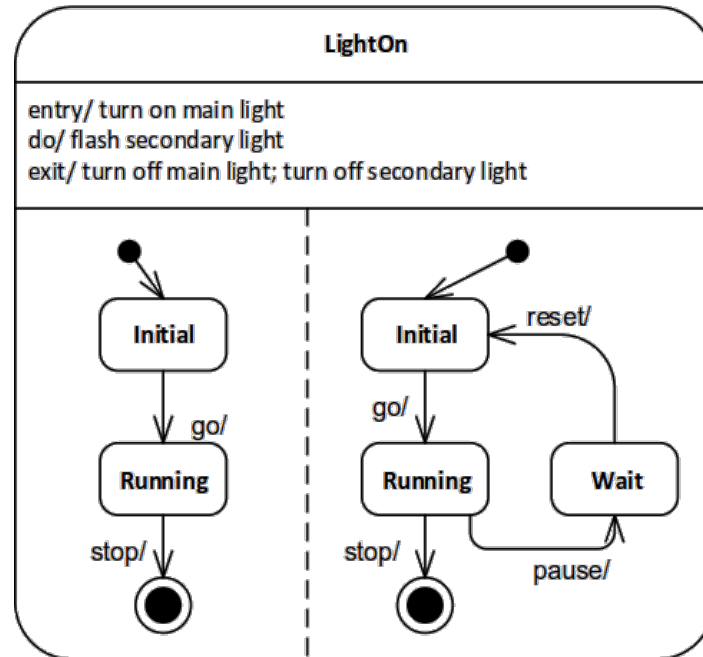


# Digital clock example

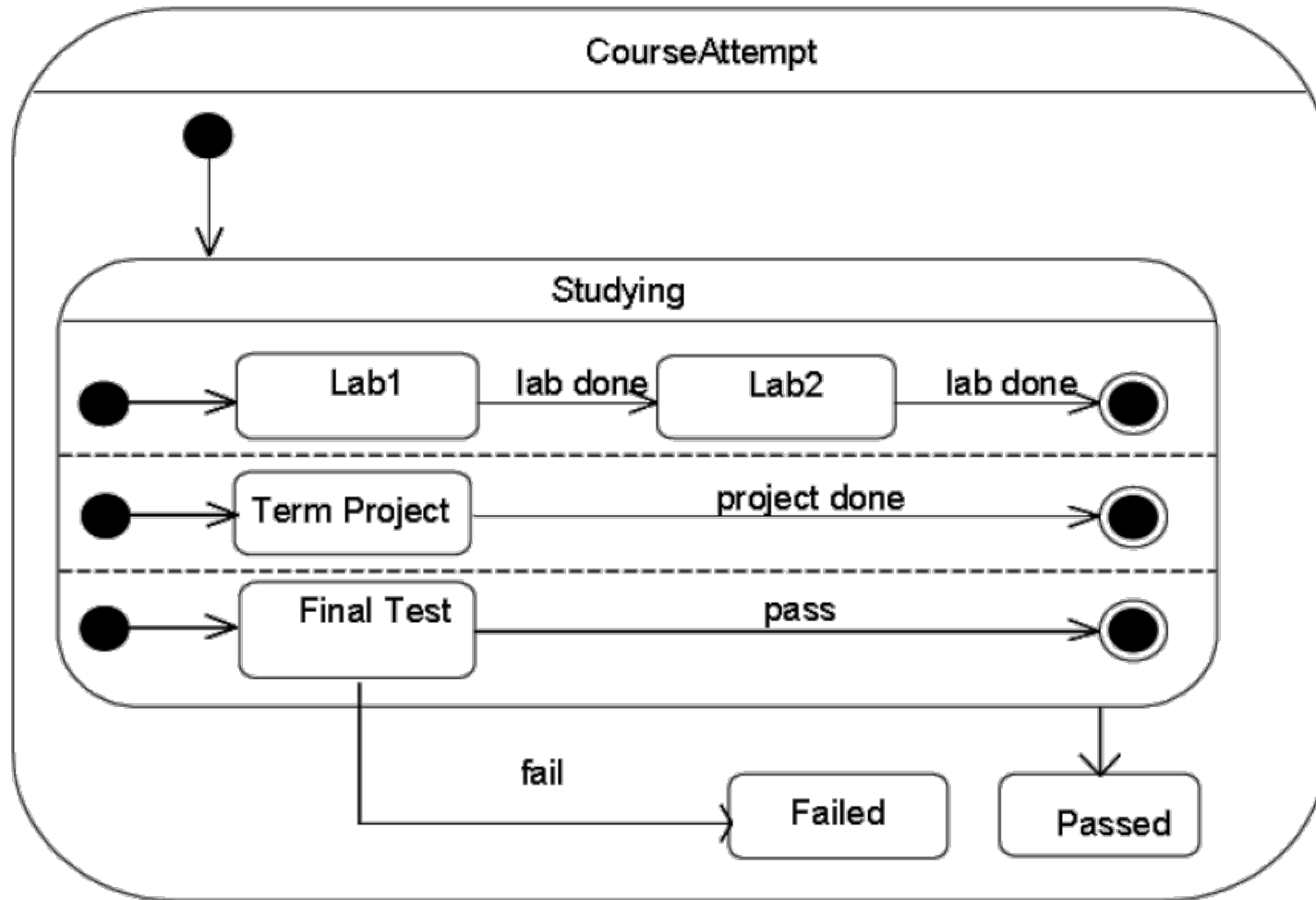


# Regions

- States and transitions can be organized in (possibly hierarchical) regions
- Orthogonal regions describe concurrent behavior

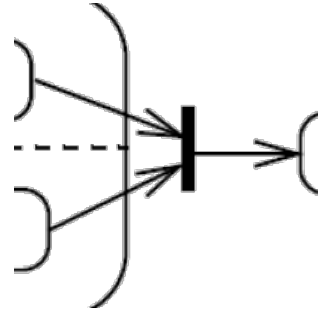


# Course attempt example



# Pseudostates

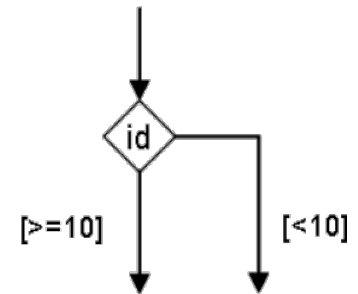
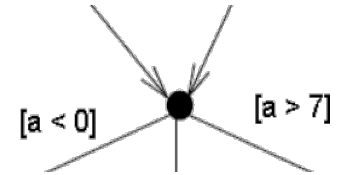
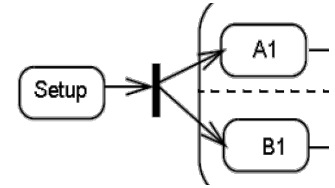
- Initial: represents the starting point of a region. It is the source of at most one transition not associated to a trigger or a guard
- Join: is the target for two or more transitions originating from vertices in different orthogonal regions; they perform a synchronization function where all incoming transitions have to complete before execution can continue through an outgoing transition





# Pseudostates

- Fork: split an incoming transition into one or more transitions terminating in vertices contained in different orthogonal regions of a composite state
- Junction: merges or splits transitions
- Choice: is a type of junction used to realize dynamic conditional branching. An else guard can be used on a predefined transition that is selected when all other guards on outgoing transitions evaluate to false

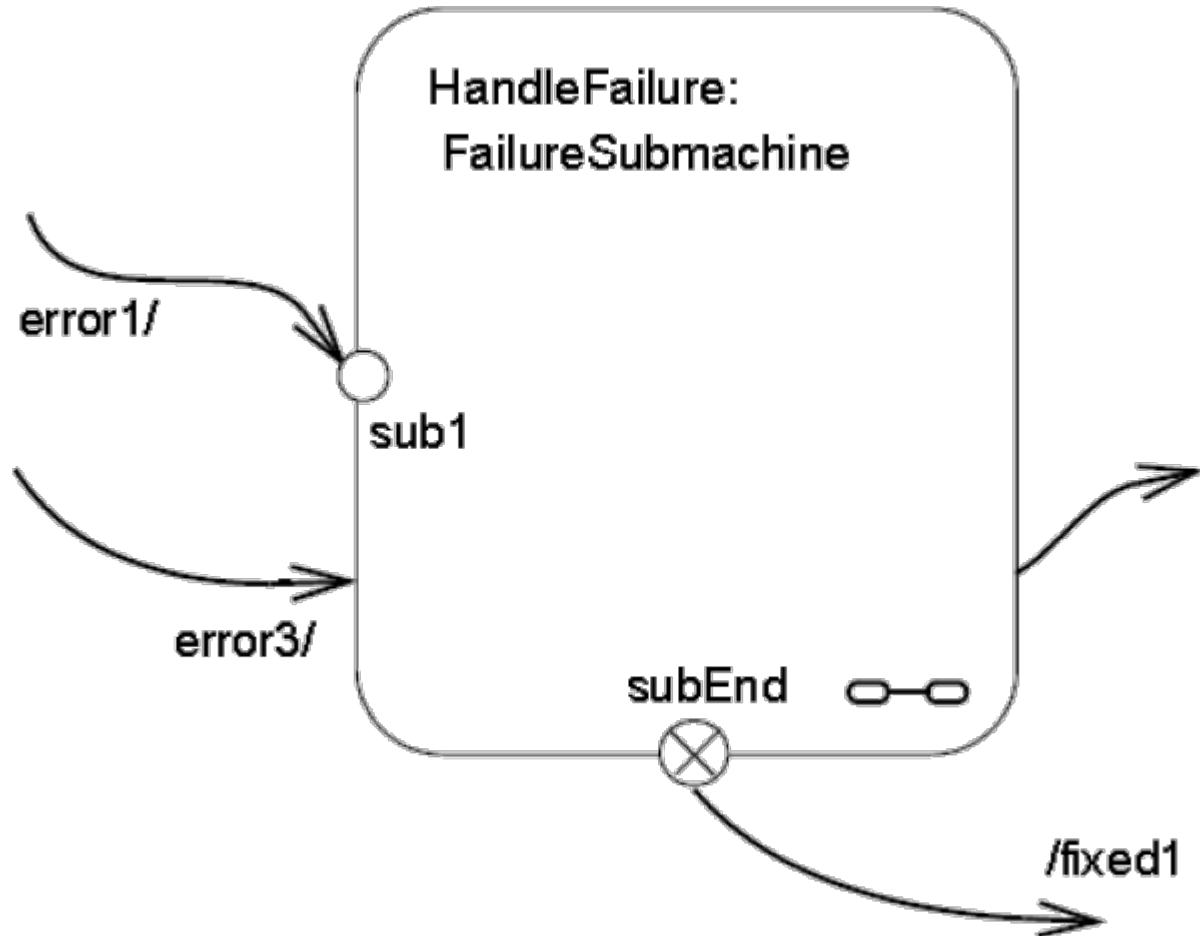


# Pseudostates

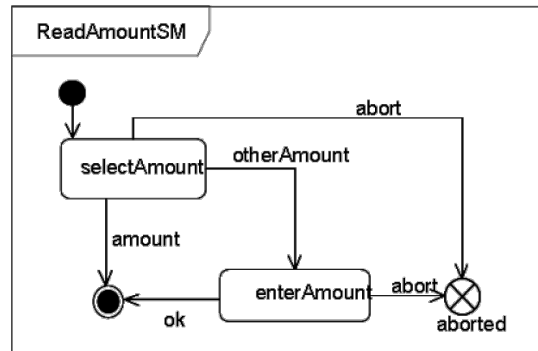
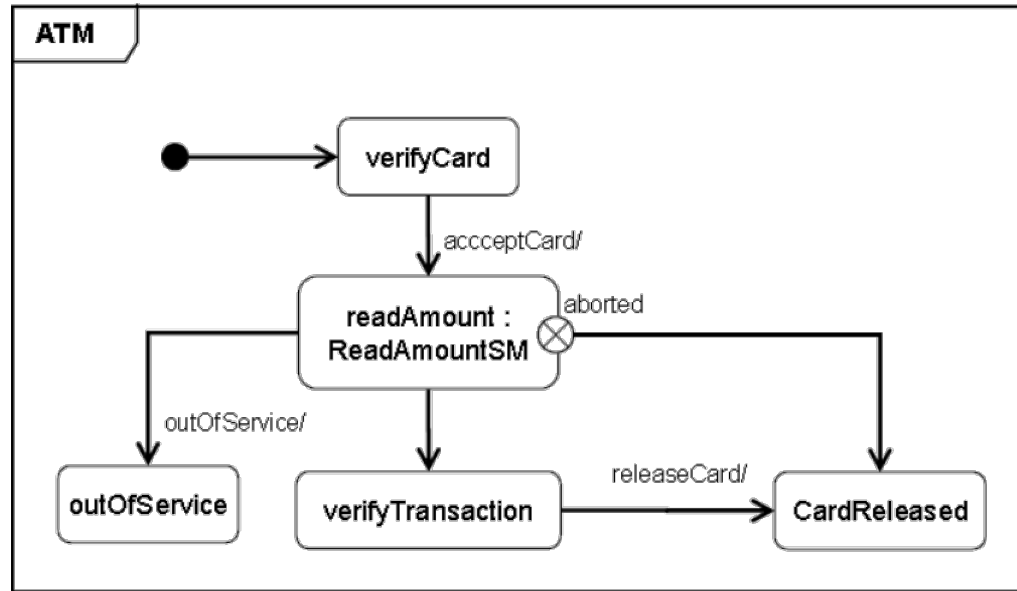
- EntryPoint: entry point for a composite or submachine state.
- ExitPoint: exit point for a composite or submachine state.
- Terminate: entering terminate the state machine execution is terminated immediately.



# Use of entry/exit points



# ATM example



# State history

State history is used to keep track of the state configuration of a region. The region can be reset to a state configuration by a (local) transition connecting to a history pseudostate.

deepHistory pseudostates restore the full state configuration; shallowHistory pseudostates restore only the topmost substate.



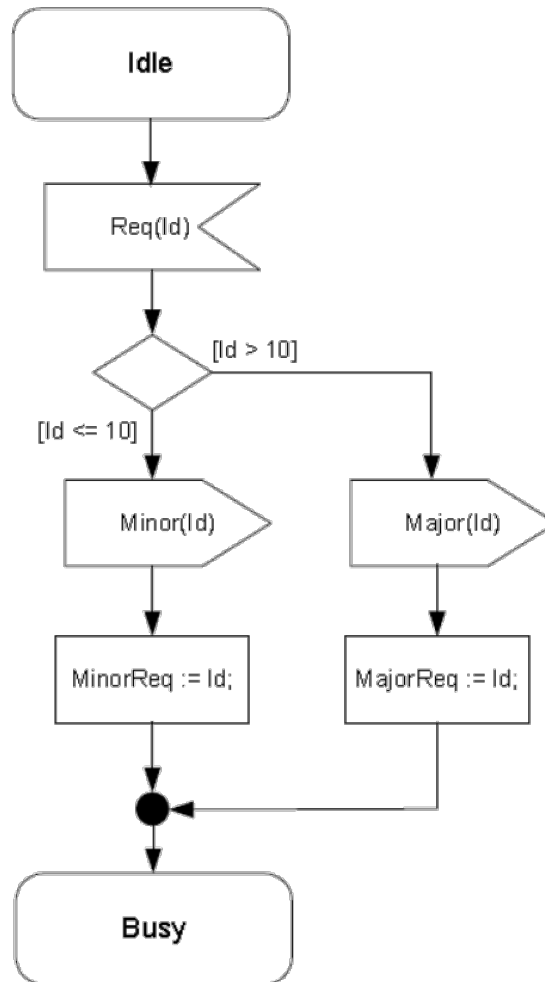
# Transitions: details

- In the course of execution a transition is:
  - reached
  - traversed
  - completed
- A transition can be associated to a set of triggers. Triggers are associated to events (event types) and are enabled when the associated event occurs

# Transitions: details

- A transition can have guard constraints. Guards are evaluated before the compound transition they are in is enabled (unless they are sourcing from a choice pseudostate). Transitions associated to guards evaluating to false are disabled
- Transitions syntax (simplified):  
**`{<trigger>}* [ '[' <guard> ' ] ' ] [ /<behavior-expr> ]`**

# Action/signal send/signal receipt symbols





# Compound transitions

A trigger can cause the traversal of an acyclic part of the state machine with no further event processing taking place. This part is called compound transition.

That means that the processing of a single event can trigger several transitions traversing several (pseudo)states.

# Run-to-completion

Upon creation a state machine performs its initialization executing the initial compound transition. Then it enters a *wait point*. When events are dispatched, triggers are evaluated and, if at least a transition can be triggered, a new compound transition is executed (a *step*) and a new wait point is reached. This cycle repeats until either the state machine complete its behavior or until it is asynchronously terminated by some external agent. This execution model is known as run-to-completion (RTC).

# Run-to-completion

Event occurrences are detected, dispatched, and processed by the StateMachine execution, one at a time. Completion event have priority, other than that event dispatching order is undefined.

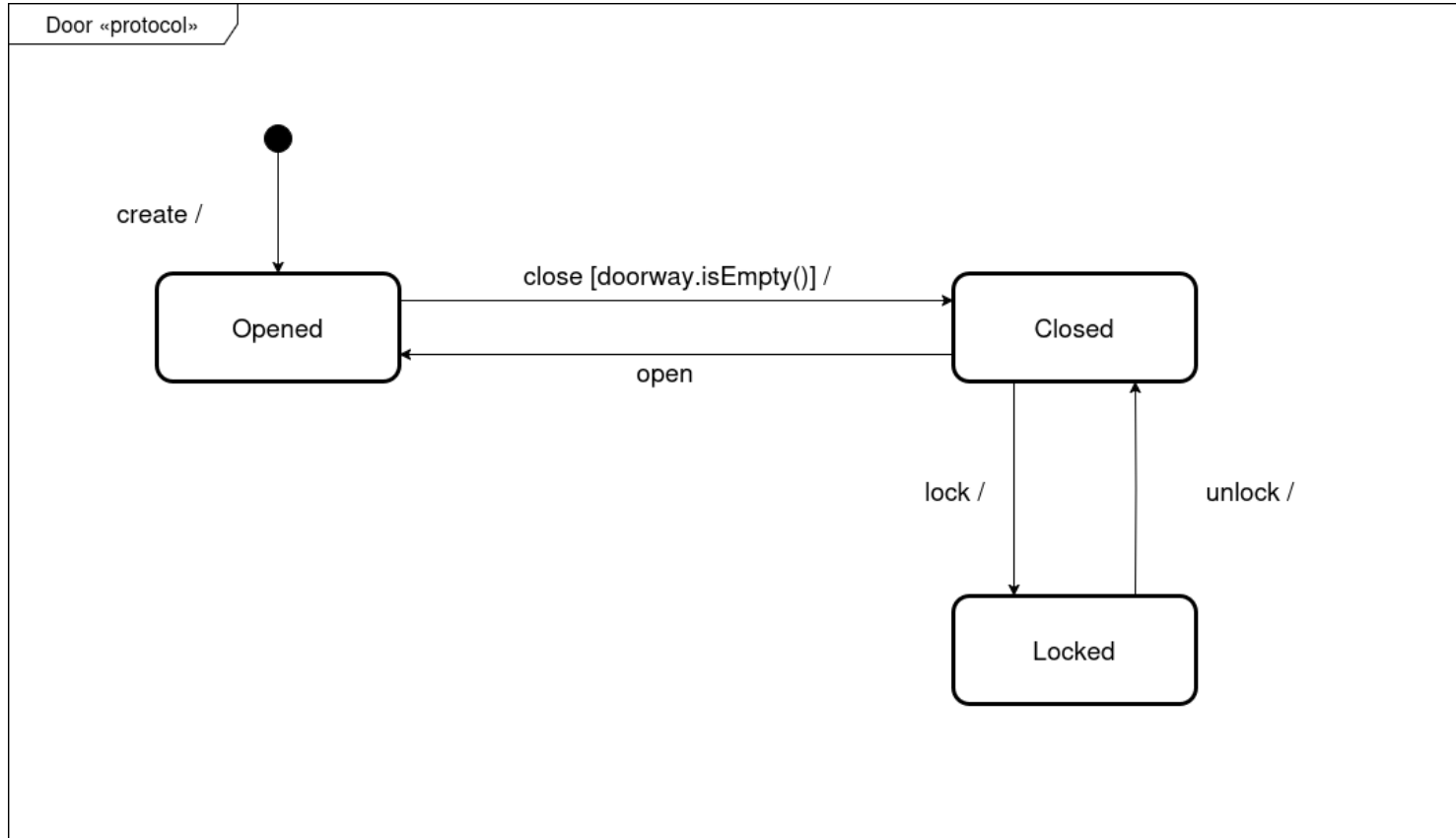
A single event can trigger multiple transitions. If these transitions have a non-empty exit states intersection they are conflicting. The priorities of conflicting Transitions are based on their relative position in the state hierarchy.

# Run-to-completion

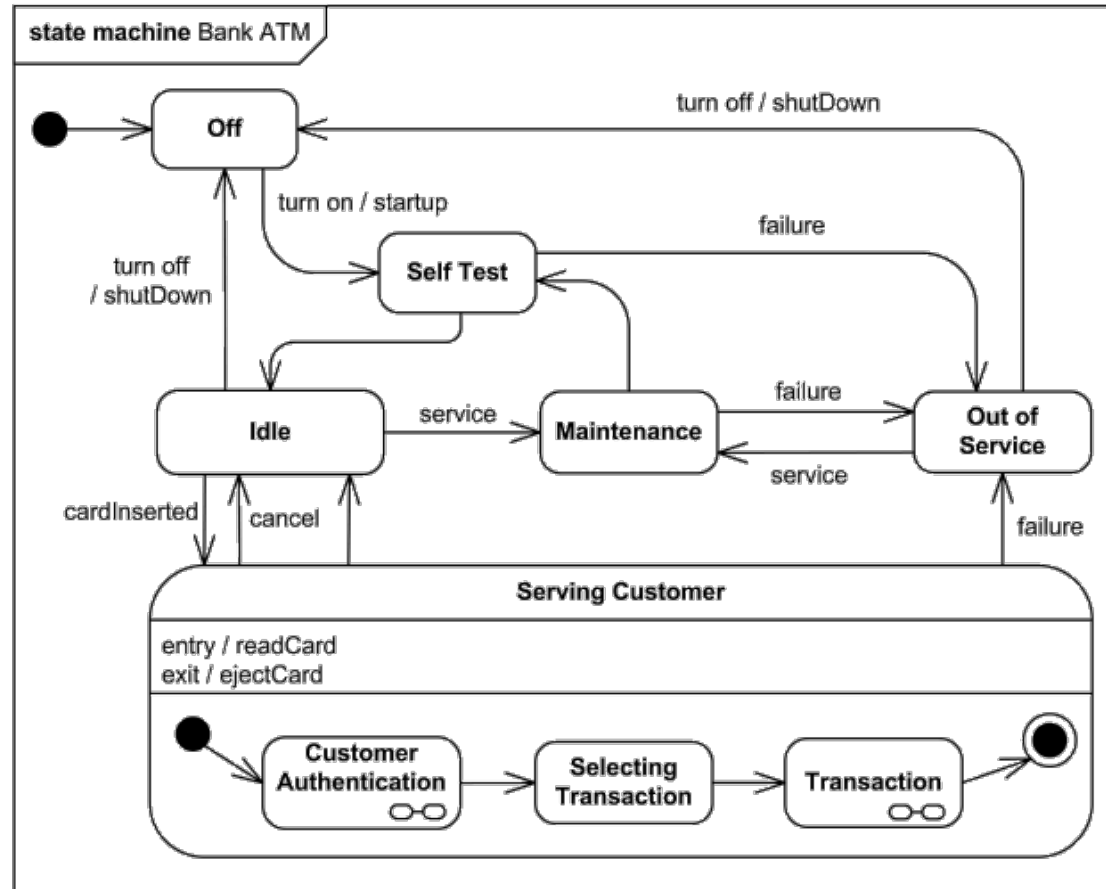
The RTC model simplifies the management of concurrency in state machines: when the machine is not in a well-defined state it is not responsive, i.e. event dispatching is not performed.

When events take place they are not immediately processed but are stored in an event queue.

# Example: door protocol



# Another ATM example



# Dishwasher example

