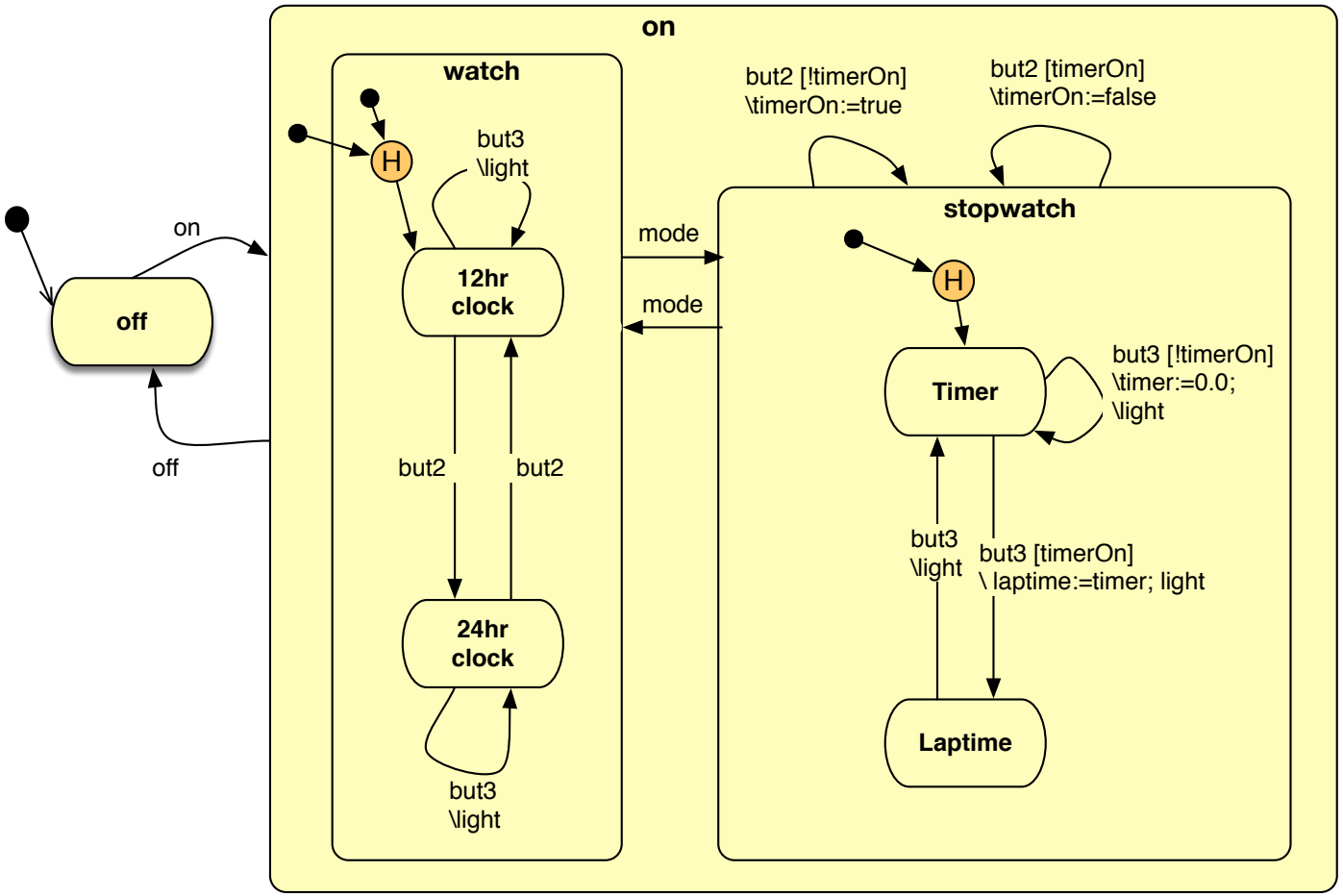
CS445 / ECE451 / CS645 / SE463

Software Requirements Specification & Analysis

Concurrent Machines

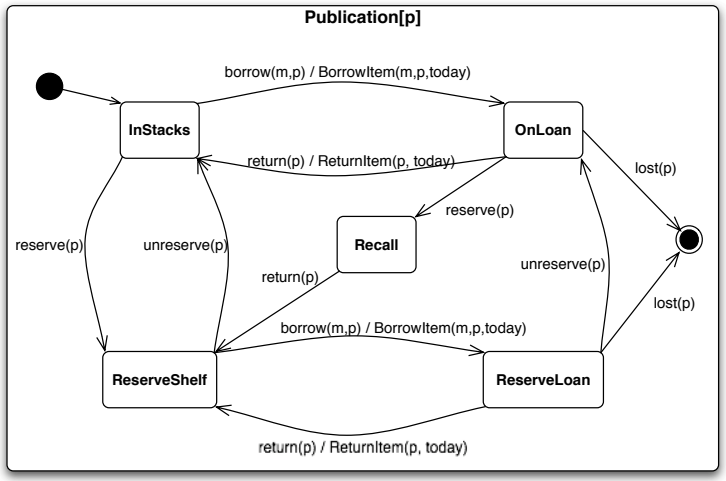
Stopwatch Example

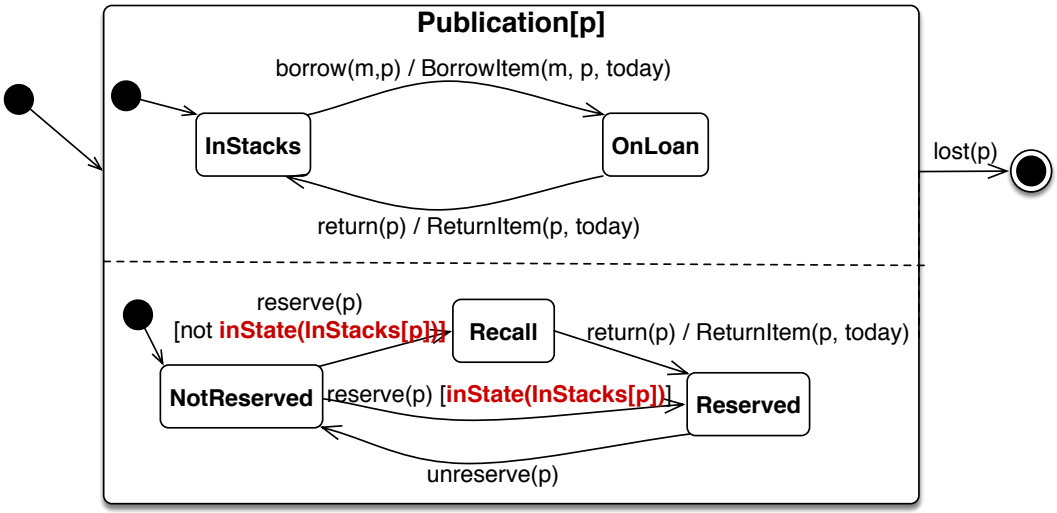


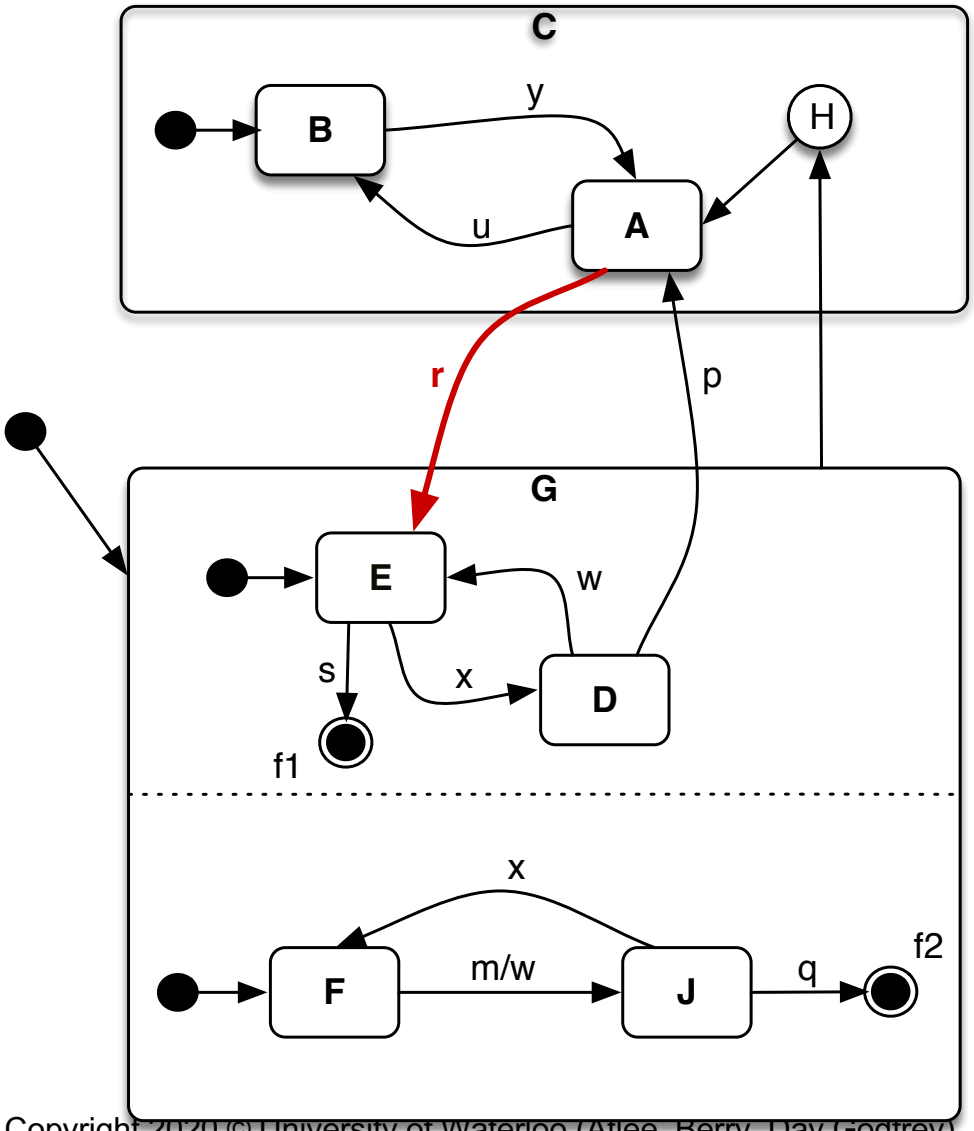
Concurrent regions

Some systems have orthogonal behaviours that are best modelled as concurrent state machines

- *Regions* within a concurrent superstate execute in parallel.
- Each has its own thread of control.
- Each can "see" and react to events /conditions in the world

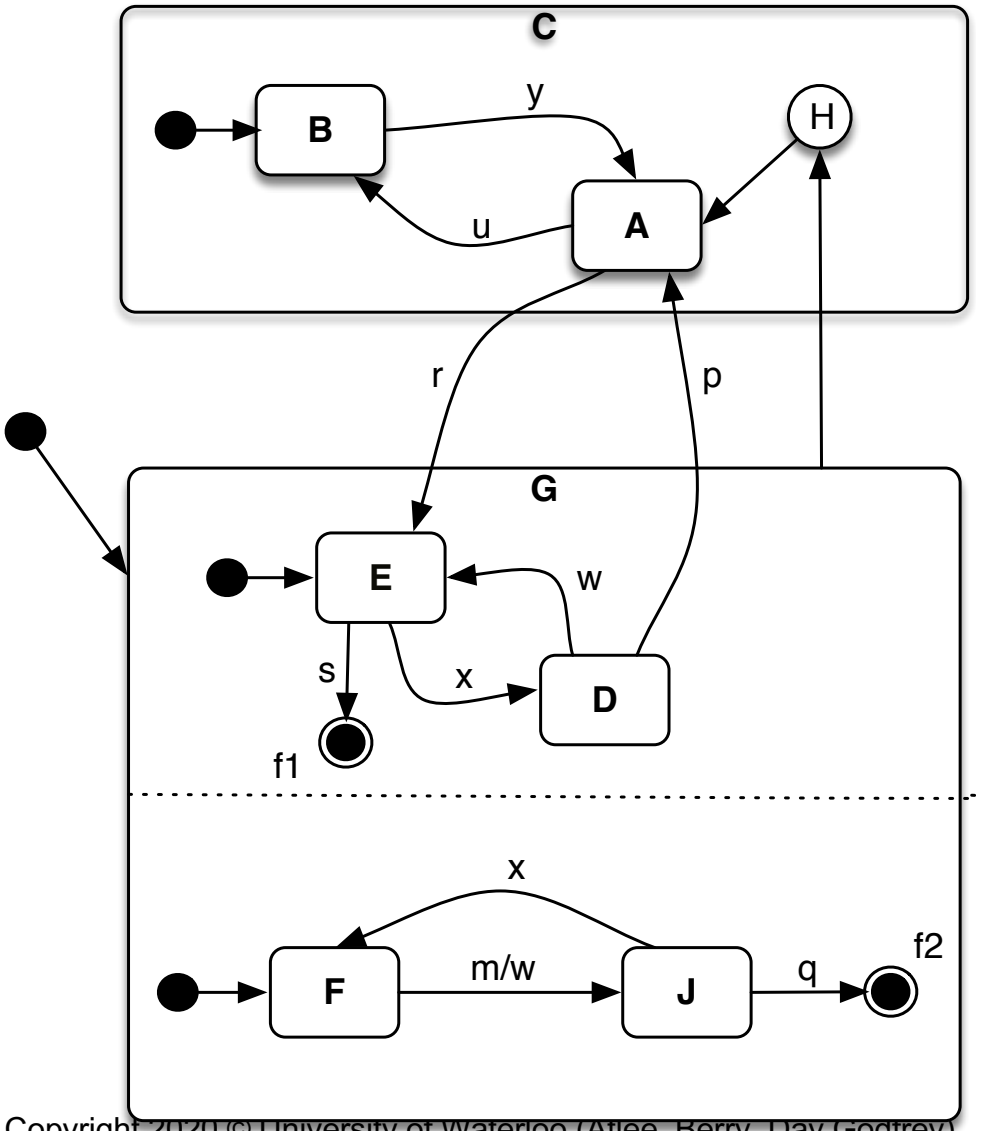






Details

- Transitions in concurrent regions triggered by the same event can execute at the same time.
- A superstate can be exited by an explicit transition out of a substate
  (All concurrent regions are also exited)
- A superstate can be entered by a transition that leads explicitly to one substate of one concurrent region.
  - The default states of all other concurrent regions are also entered
  - History + deep history can be used as destination states in regions



Exercise

- What is the initial state of the system?

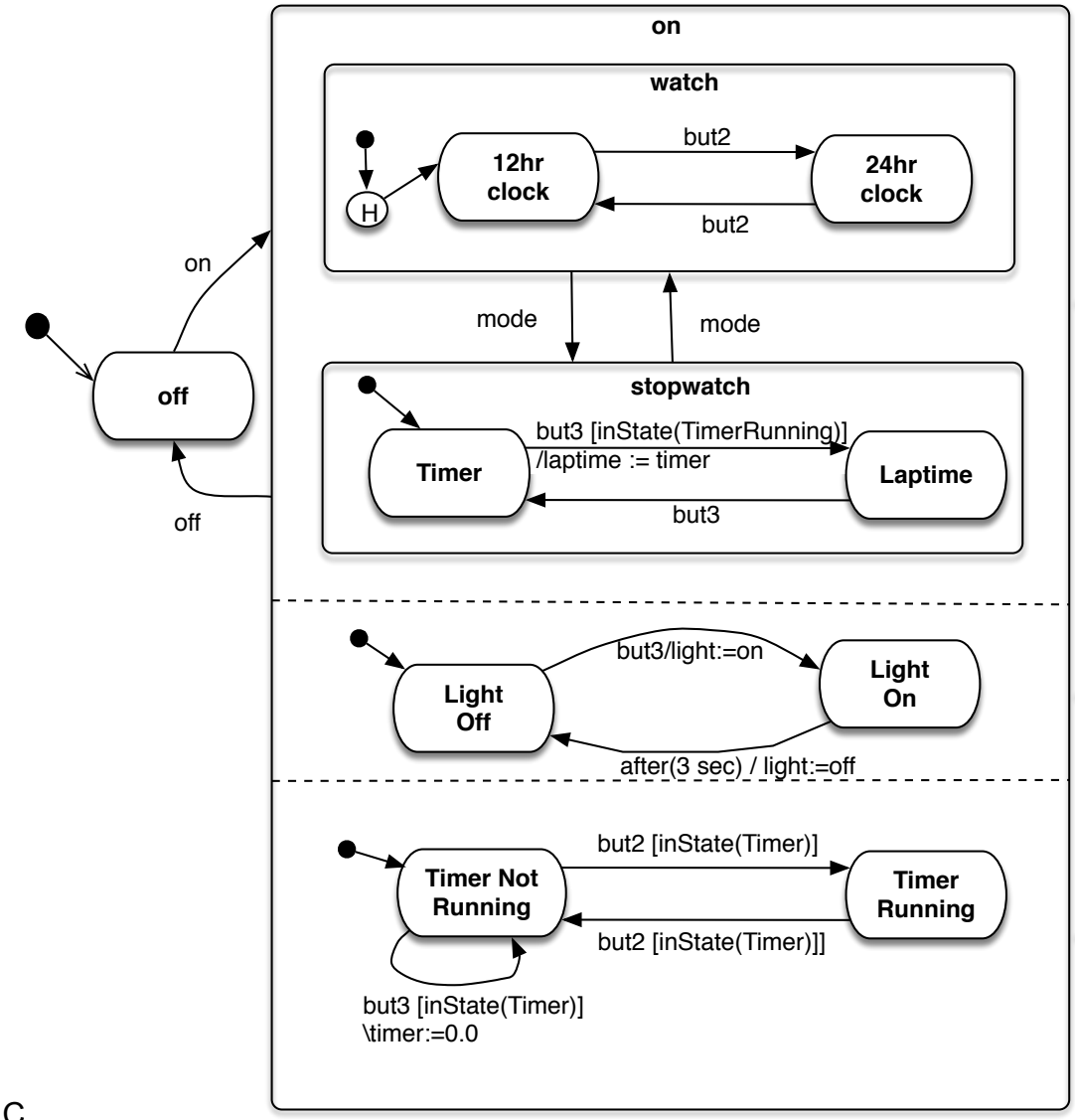
  E/F

- What is the state of the system after the following sequence of input events:

  x p r m s q

E/F→D/F → A → E/F → EJ → f1/J → f1/f2 → A

Stopwatch



Event Macros

on = when [On/Off:Button.pressed]
off = when [On/Off:Button.pressed]
mode = when [mode:Button.pressed]
but2 = when [but2:Button.pressed]
but3 = when [but3:Button.pressed]

Variable Macros

light = display.light
timer = watch.timer
laptime = watch.laptime

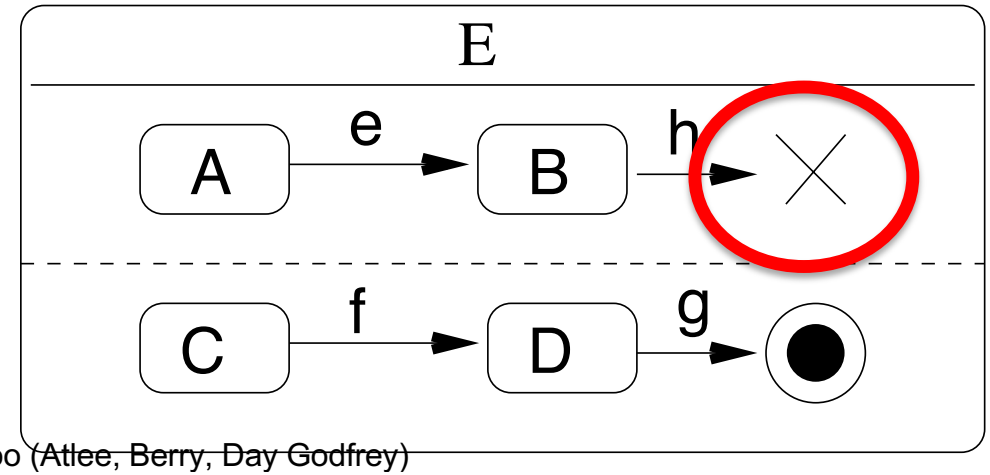
Vars

~~timerOn: bool~~

Final and Termination States

To represent the end of system execution, use the termination state.

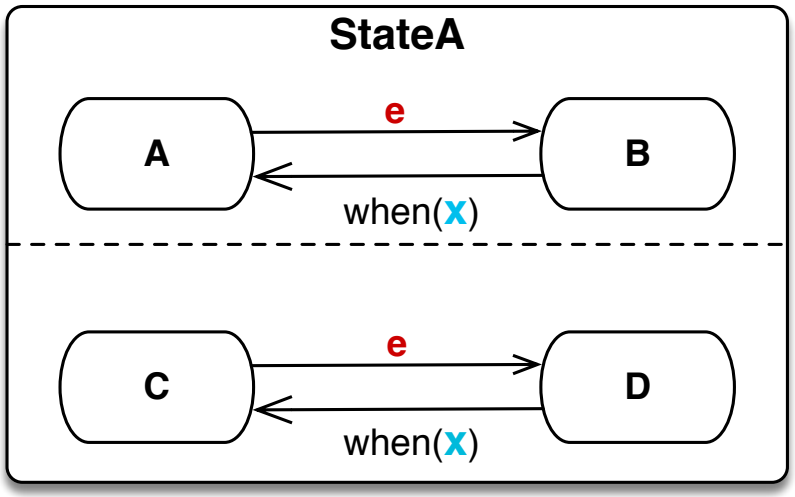
- It's like a poison pill for the machine
- Not the same as a final state in a region, which waits patiently for the other regions to finish



Coordination of Regions

Ways to coordinate regions with each other

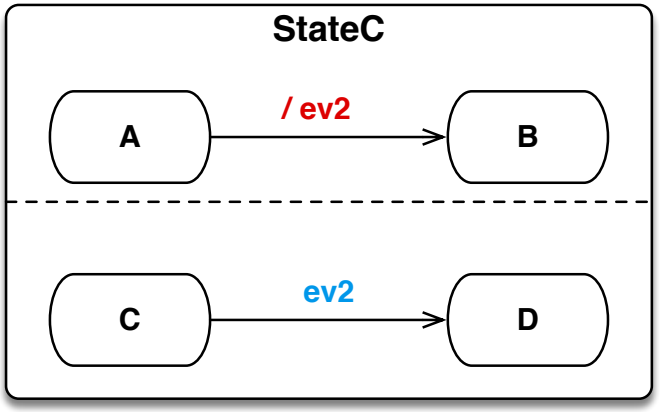
1. Regions react to the same input event
2. Regions react to the value of the same «interface» phenomenon



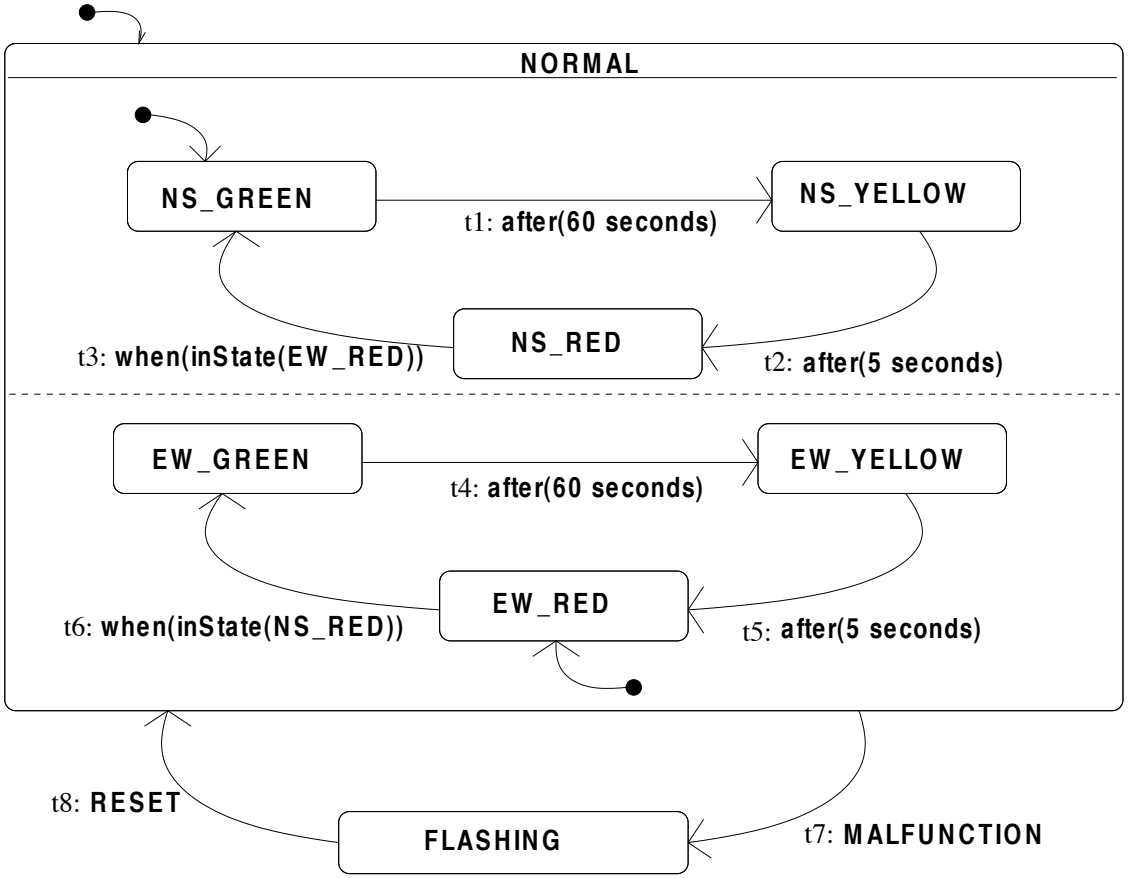
Coordination of Regions

Ways to coordinate regions with each other

1. Regions react to the same input event
2. Regions react to the value of the same ≪interface≫ phenomenon
3. One region generates an event that other regions react to
4. A region's transition has guard [inState(x)] that evaluates to *true* iff another region is in state *x*



Traffic light example



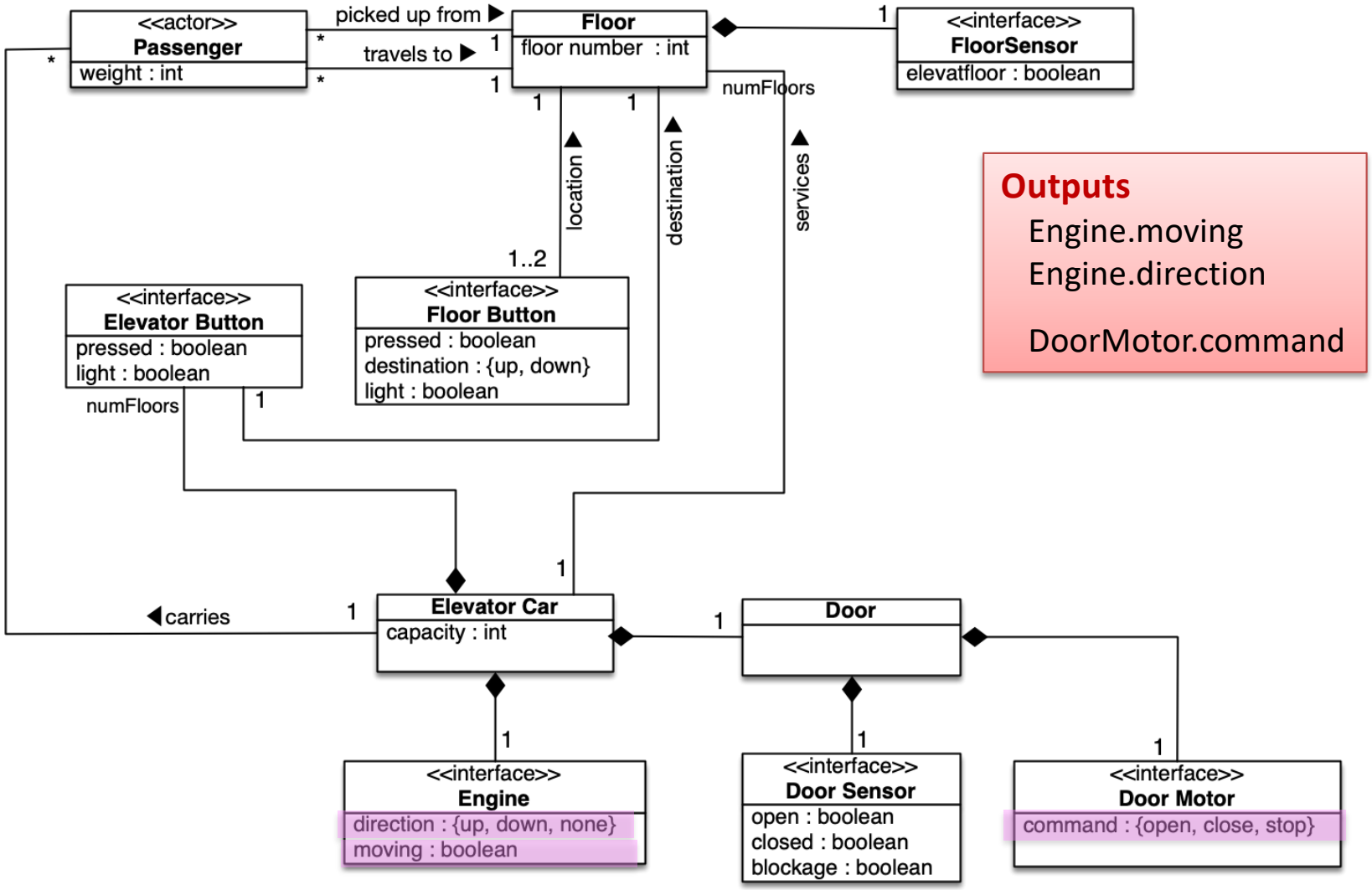
Creating a Behaviour Model

1. Identify input and output events
2. Think of a natural partitioning into states
   - *Activity states* – system performs activity or operation
   - *Idle states* – system waits for input
   - *System modes* – use different states to distinguish between different reactions to an event
3. Consider the behaviour of the system for each input at each state.
4. Revise (using hierarchy, concurrency, state events, history)

Example: Elevator

- An elevator passenger wants to travel from one floor to a higher (lower) floor, and presses the Up (Down) button
- The light beside the button must then be lit, if it was not lit before.
- The elevator must arrive reasonably soon, travelling in an upwards direction.
- The direction of travel is indicated by an arrow illuminated when the elevator arrives.
- The doors must open and stay open long enough for the passenger to enter the elevator.
- The doors must never be open except when the elevator is stationary at a floor.

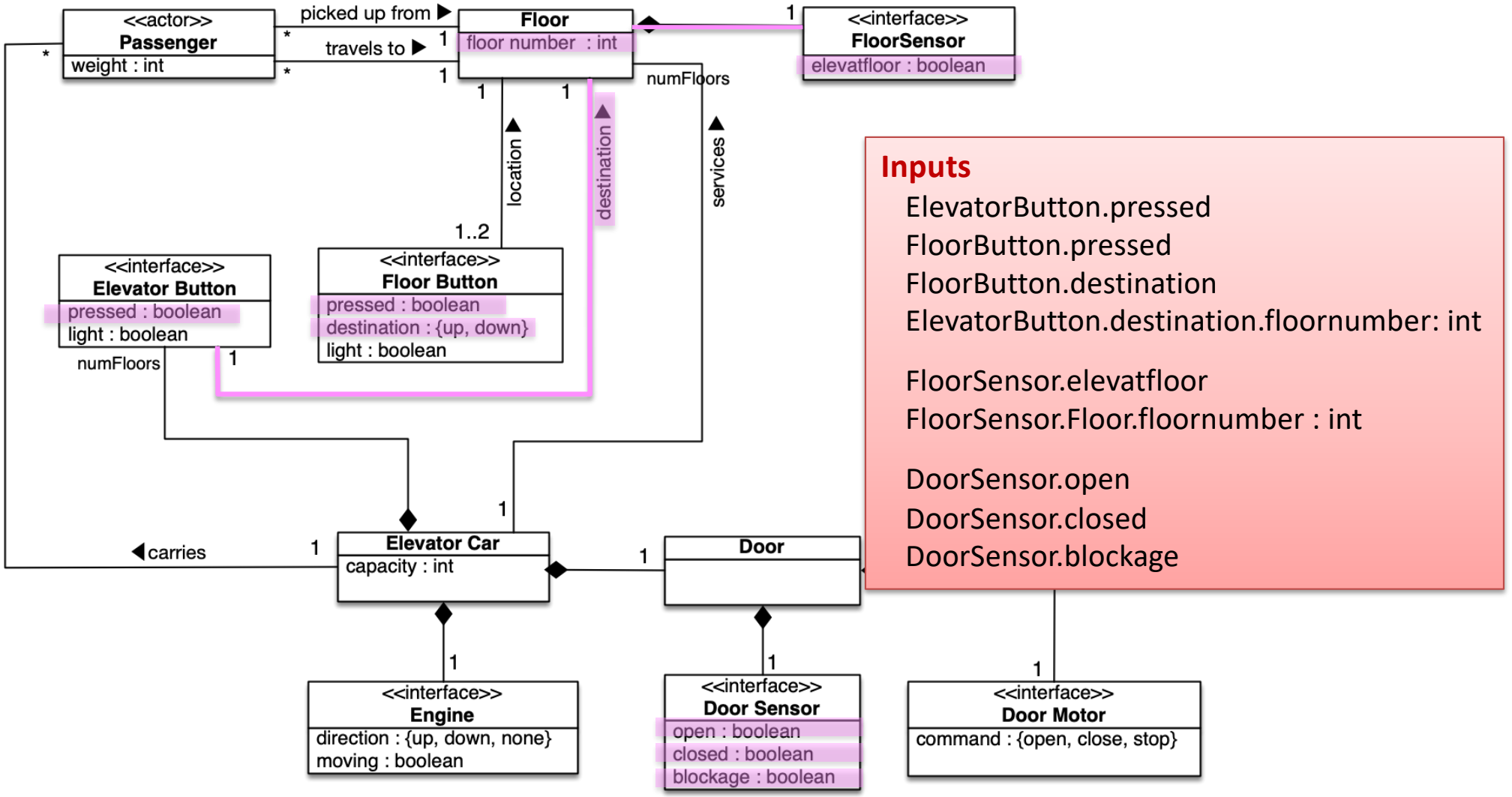
Michael Jackson, *Software Requirements and Specifications, Addison-Wesley, 1995.*

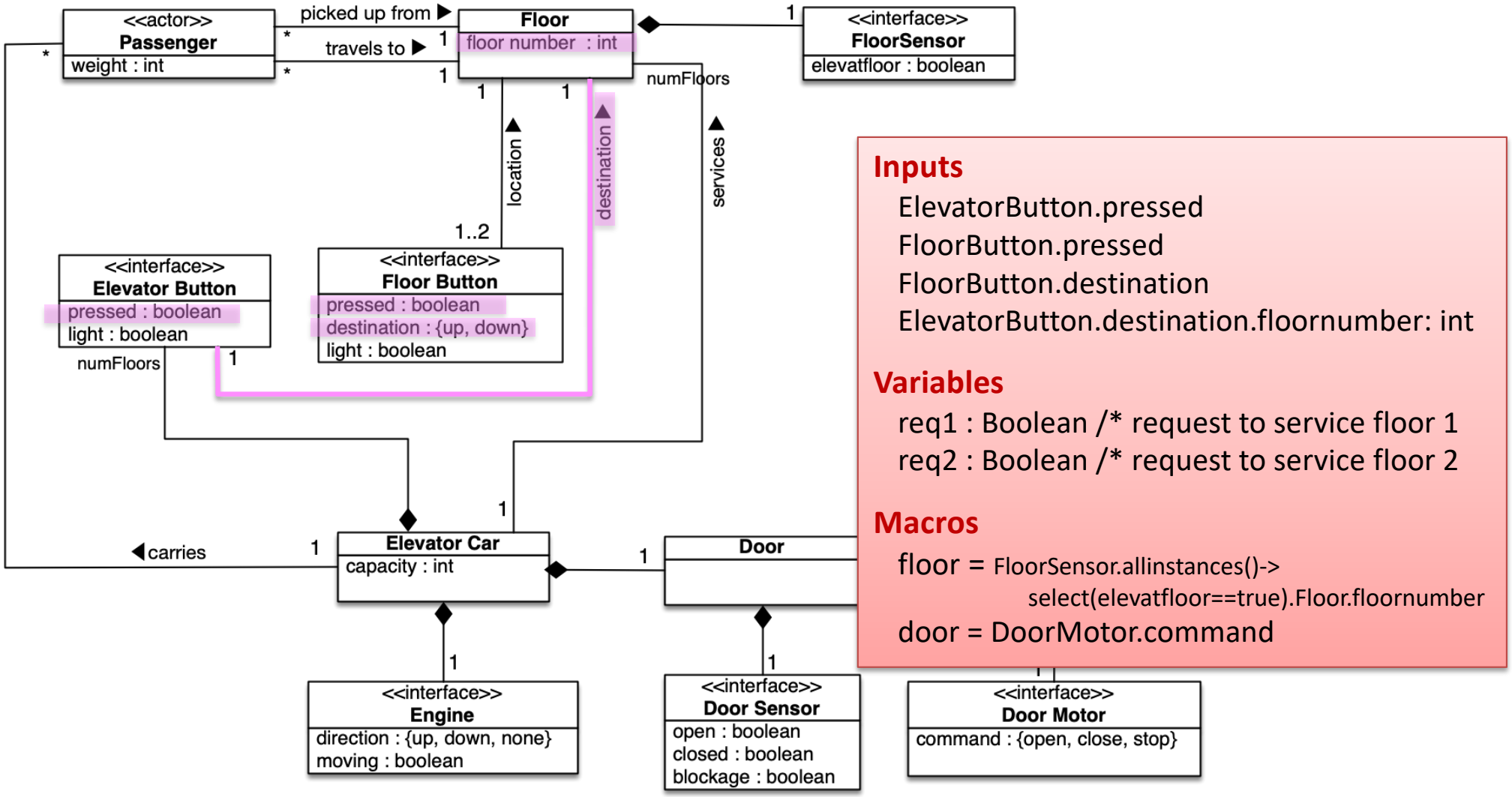


Outputs

Engine.moving
Engine.direction

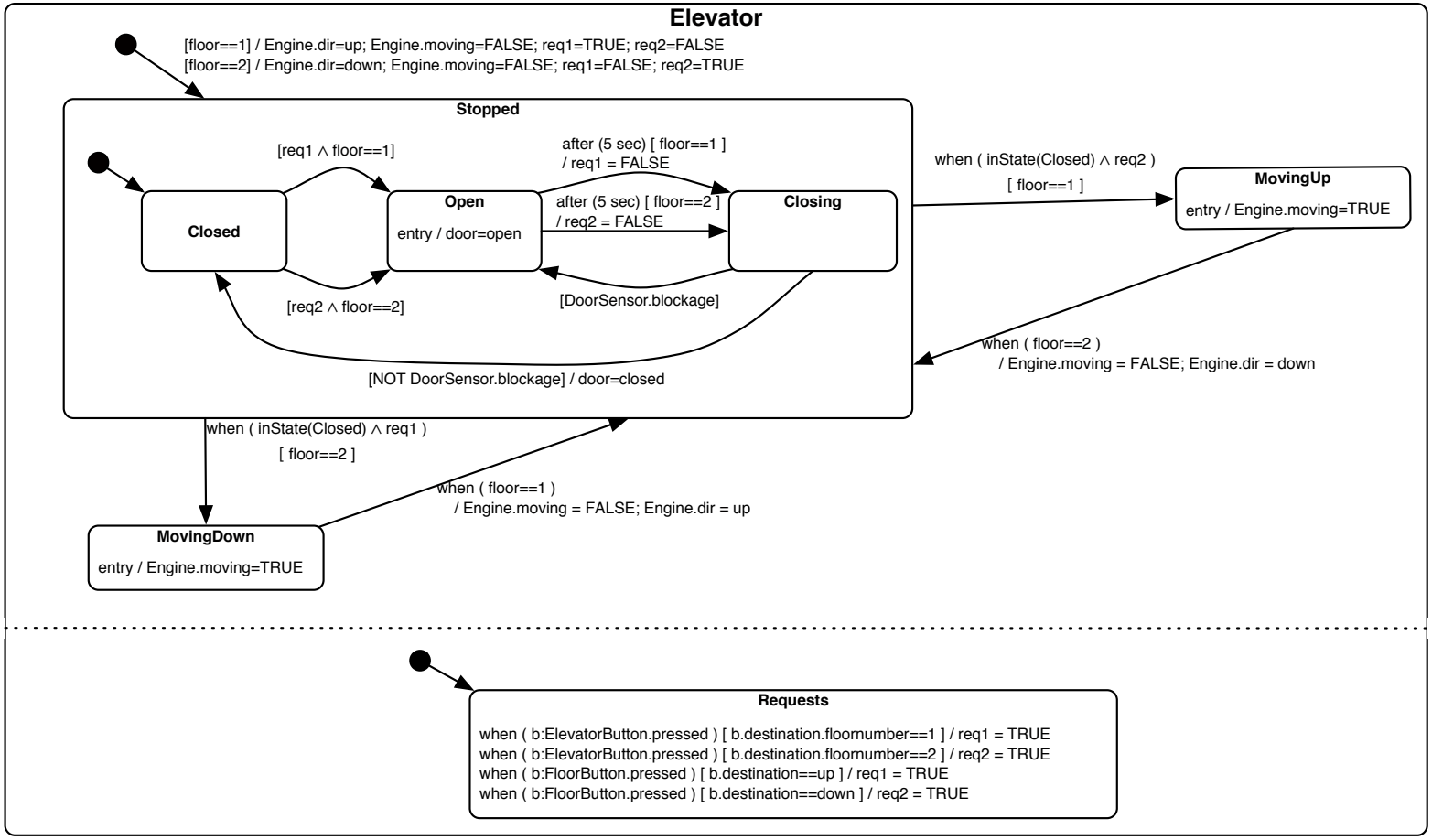
DoorMotor.command





VARIABLES
req1 : boolean := false (* outstanding request for floor1 *)
req2 : boolean := false (* outstanding request for floor2 *)

MACROS
floor = FloorSensor.allinstances()-> select(elevatfloor==true).Floor.floornumber
door = DoorMotor.command



Validating Behaviour Models

- Avoid inconsistency: multiple transitions that leave the same state under the same event/conditions.
- Ensure completeness: specify a reaction for every possible input at a state.
- Walkthrough: compare the behaviour of your state diagrams with the use-case scenarios.
  - All paths in scenarios should be paths in the state machines

Good style

Aim for Clarity

- Use states to model modes of operation, and use variables to model other information that affect flow of control
- Fewer, simpler transitions are better
  - Use hierarchy to combine similar transitions
  - Use concurrency to separate orthogonal aspects of the problem (monitoring vs. controlling the environment)
  - Use hierarchy, and entry/exit actions, to abbreviate common behaviour

References

Craig Larman, *Applying UML and Patterns, 3ed.,* Prentice Hall, 2004.

- Chapter 23: “UML State Machine Diagrams and Modeling”

Lenny Delligatti, *SysML Distilled: A Brief Guide to the Systems Modeling Language*, Addison-Wesley Professional, 2013.

- Chapter 8: “State Machine Diagrams”