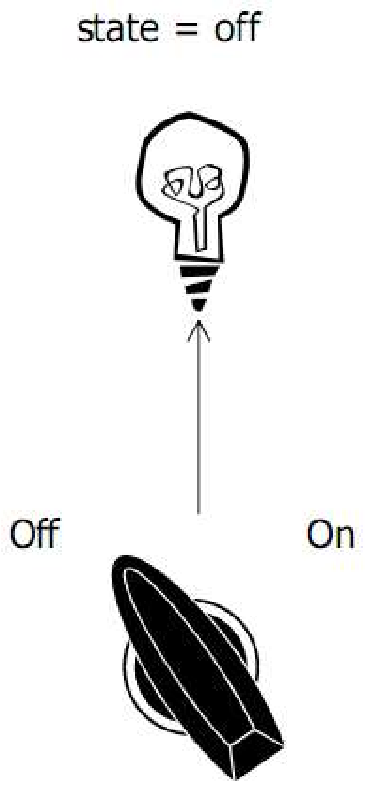
Behavioral (Dynamic) Modeling

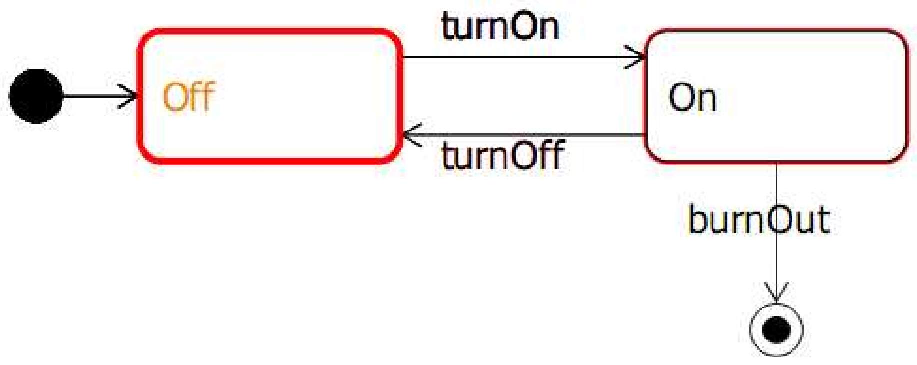
- Behavioral (Dynamic) Modeling
  - Use Case Diagrams (Already covered)
  - Activity Diagrams (Already covered)
  - Interaction Diagrams (Already covered)
    - Sequence Diagram
    - Communication Diagram
  - Statechart diagram OR State machine diagram OR State Transition diagram OR state diagram
    - Different names

Statechart Diagrams

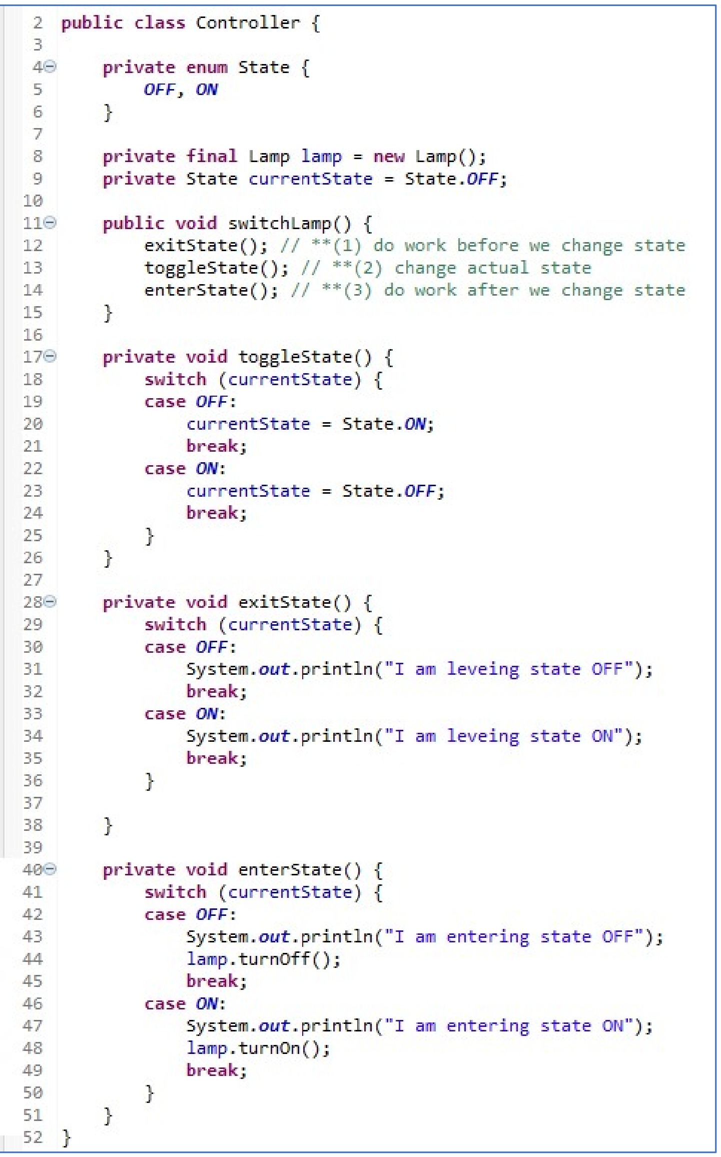
- State machine diagrams capture the behavior of a software system.
- State machines can be used to model the behavior of a class, subsystem or an entire application
- They also provide a way of modeling communications that occur with external entities via a protocol or event-based system

State Machine Example

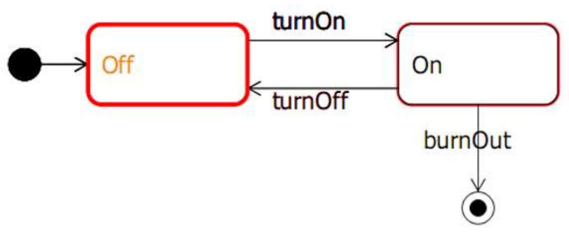


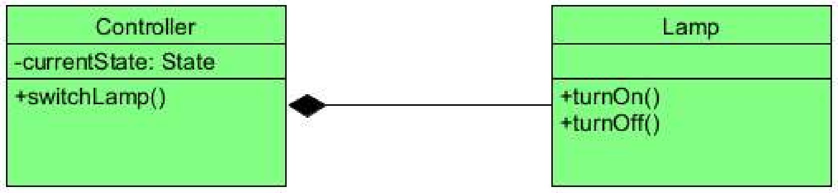


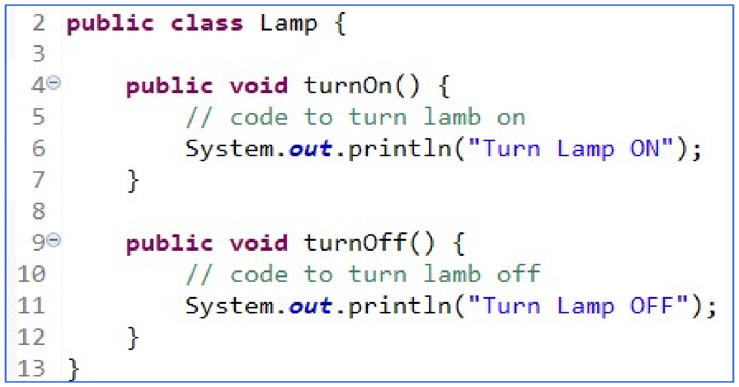
State Machine Implementation Example



```java
public class Controller {

    private enum State {
        OFF, ON
    }

    private final Lamp lamp = new Lamp();
    private State currentState = State.OFF;

    public void switchLamp() {
        exitState(); // **(1) do work before we change state
        toggleState(); // **(2) change actual state
        enterState(); // **(3) do work after we change state
    }

    private void toggleState() {
        switch (currentState) {
        case OFF:
            currentState = State.ON;
            break;
        case ON:
            currentState = State.OFF;
            break;
        }
    }

    private void exitState() {
        switch (currentState) {
        case OFF:
            System.out.println("I am leveing state OFF");
            break;
        case ON:
            System.out.println("I am leveing state ON");
            break;
        }

    }

    private void enterState() {
        switch (currentState) {
        case OFF:
            System.out.println("I am entering state OFF");
            lamp.turnOff();
            break;
        case ON:
            System.out.println("I am entering state ON");
            lamp.turnOn();
            break;
        }
    }
}
```

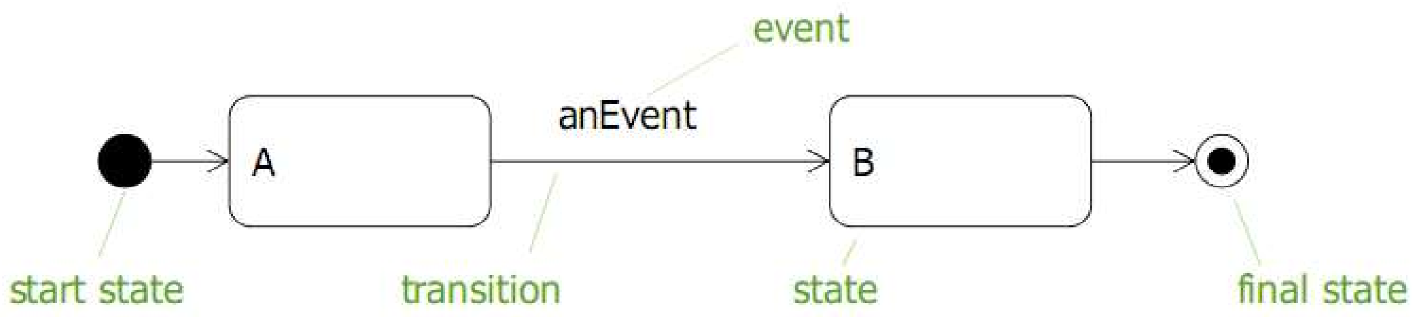






```java
public class Lamp {

    public void turnOn() {
        // code to turn lamb on
        System.out.println("Turn Lamp ON");
    }

    public void turnOff() {
        // code to turn lamb off
        System.out.println("Turn Lamp OFF");
    }
}
```

Basic State Machine Syntax



- Every state machine should have a *start state* which indicates the first state of the sequence
- Unless the states cycle endlessly, state machines should have a *final state* which terminates the sequence of transitions

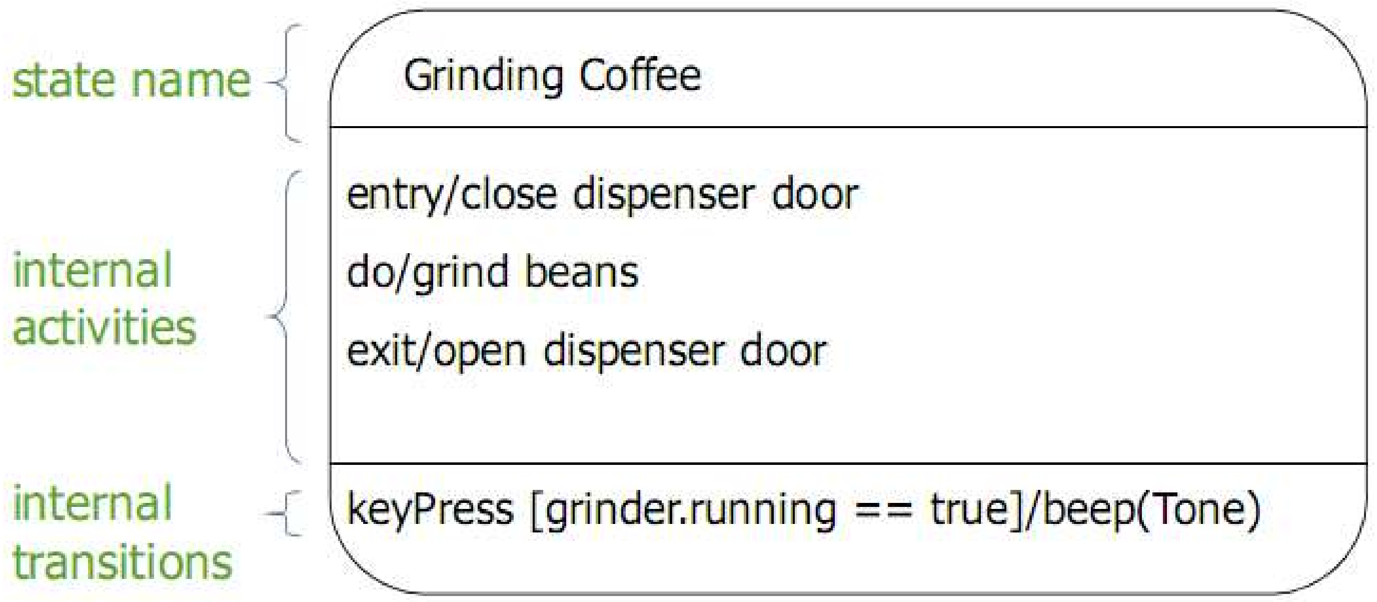
Outline

- Statechart Diagrams
  - Behavioral State Machines
    - States
    - Transitions
    - Composite States
    - Pseudostates
    - Submachine States

States

- "A condition or situation during the life of an object during which it satisfies some condition, performs some activity or waits for some event"
- The state of an object at any point in time is determined by:
  - The values of its attributes
  - The relationships it has to other objects
  - The activities it is performing

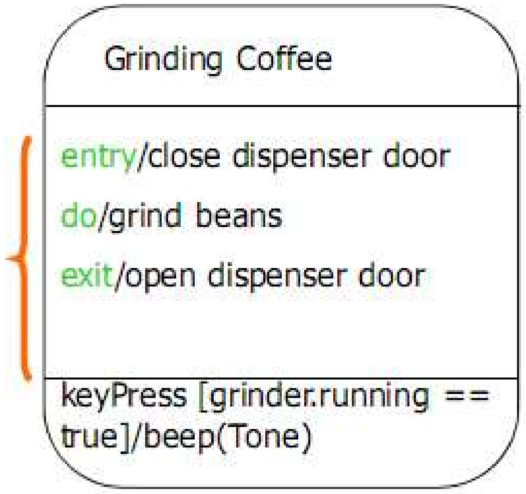
State Syntax



State activities

UML offers three labels:

- **Entry**: triggers when a state is entered. The entry action executes before anything else happens in the state
- **Do**: Executes as long as a state is active. The do activity executes after the entry activity and can run until it completes, or as long as the state machine is in this state
- **Exit**: triggers when leaving a state. The exit action executes as the last thing in the state before a transition occurs



Grinding Coffee
entry/close dispenser door
do/grind beans
exit/open dispenser door
keyPress [grinder.running == true]/beep(Tone)

Internal transitions

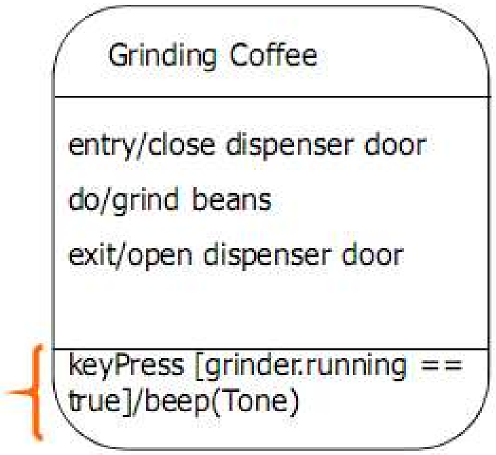
- Internal transitions occur within the state. They do not transition to a new state

Syntax: trigger [gard] /effect

trigger: what may cause a transition to occur. Typically an event

guard: boolean expression, evaluated before a transition is fired

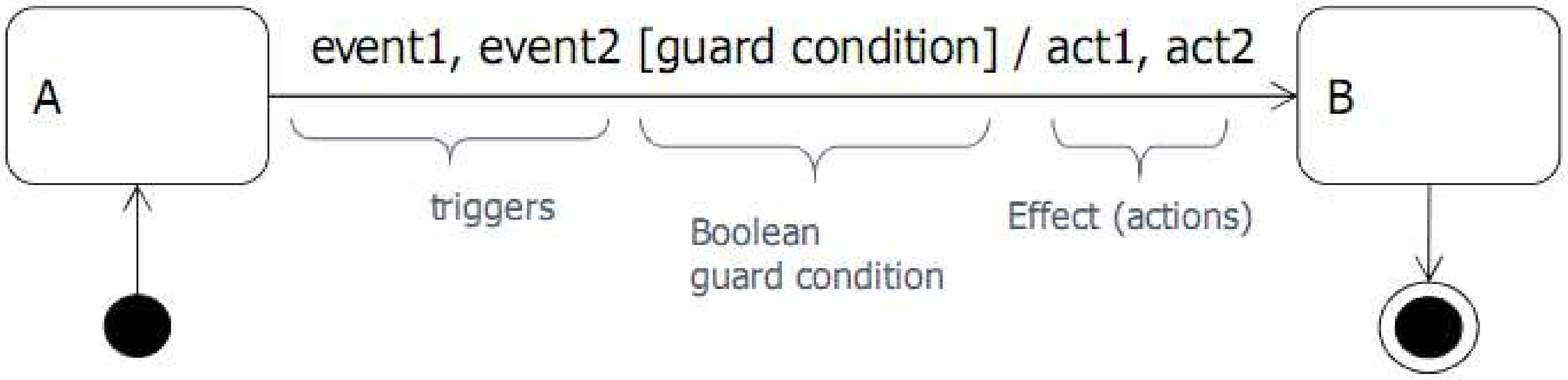
effect: activity that is executed when a transition happens


Grinding Coffee
entry/close dispenser door
do/grind beans
exit/open dispenser door
keyPress [grinder.running == true]/beep(Tone)

Outline

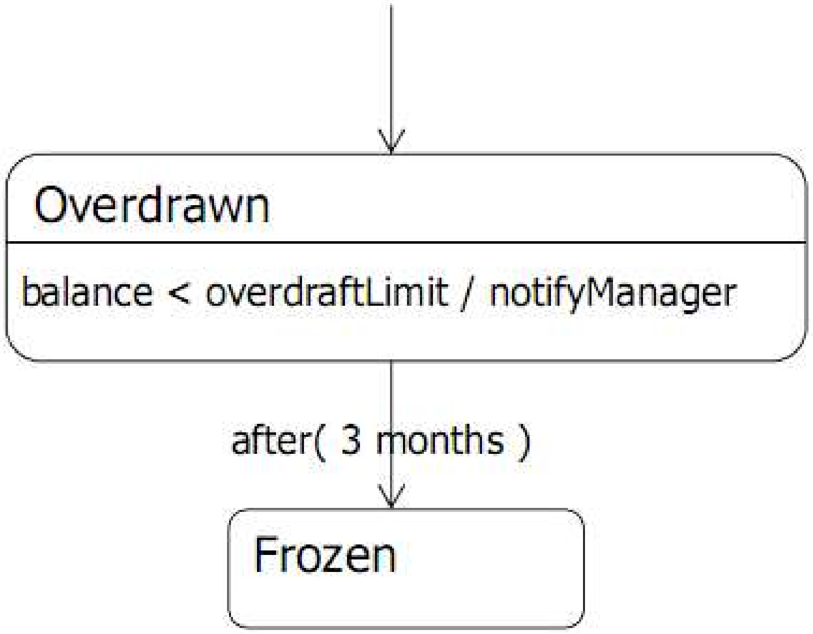
- Statechart Diagrams
  - Behavioral State Machines
    - States
    - Transitions
    - Composite States
    - Pseudostates
    - Submachine States

Transitions



Time Events

- Time events occur when a time expression becomes true
- There are two keywords, after and when
- Elapsed time:
  - after(3 months)
- Absolute time:
  - when(date =20/3/2010)



State Types

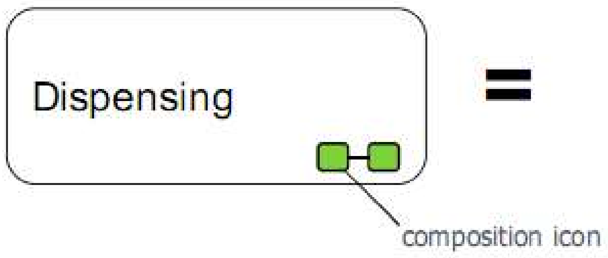
- UML defines three types of states
  - **Simple states**: have no substates
  - **Composite states**: have one or more regions (containers) for substates. Can be either *simple (Or-state)* or *orthogonal (And-state)* substates
  - **Submachine states**: semantically equivalent to composite states, submachine states have substates that are contained within a *substate machine.* Intended to group states for reuse

Outline

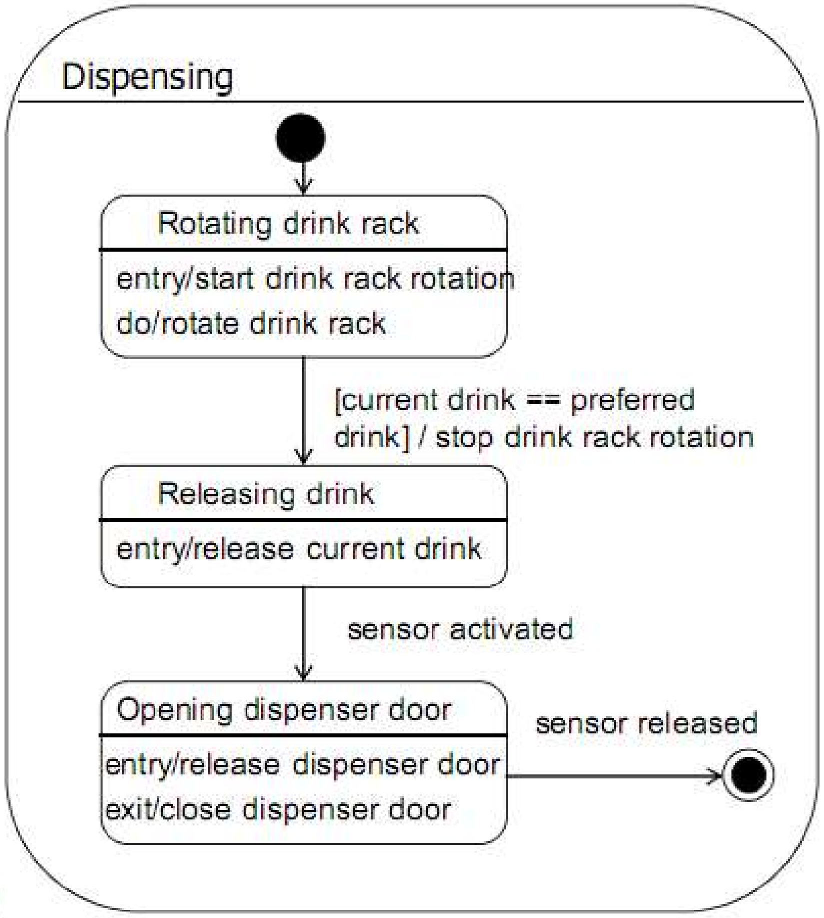
- Statechart Diagrams
  - Behavioral State Machines
    - States
    - Transitions
    - Composite States
    - Pseudostates
    - Submachine States

State Types: Composite States

- Simple composite state (*OR-state*)
  - Exactly one region
  - Allows encapsulation
- A *decomposition compartment* is a detailed view of the composite state

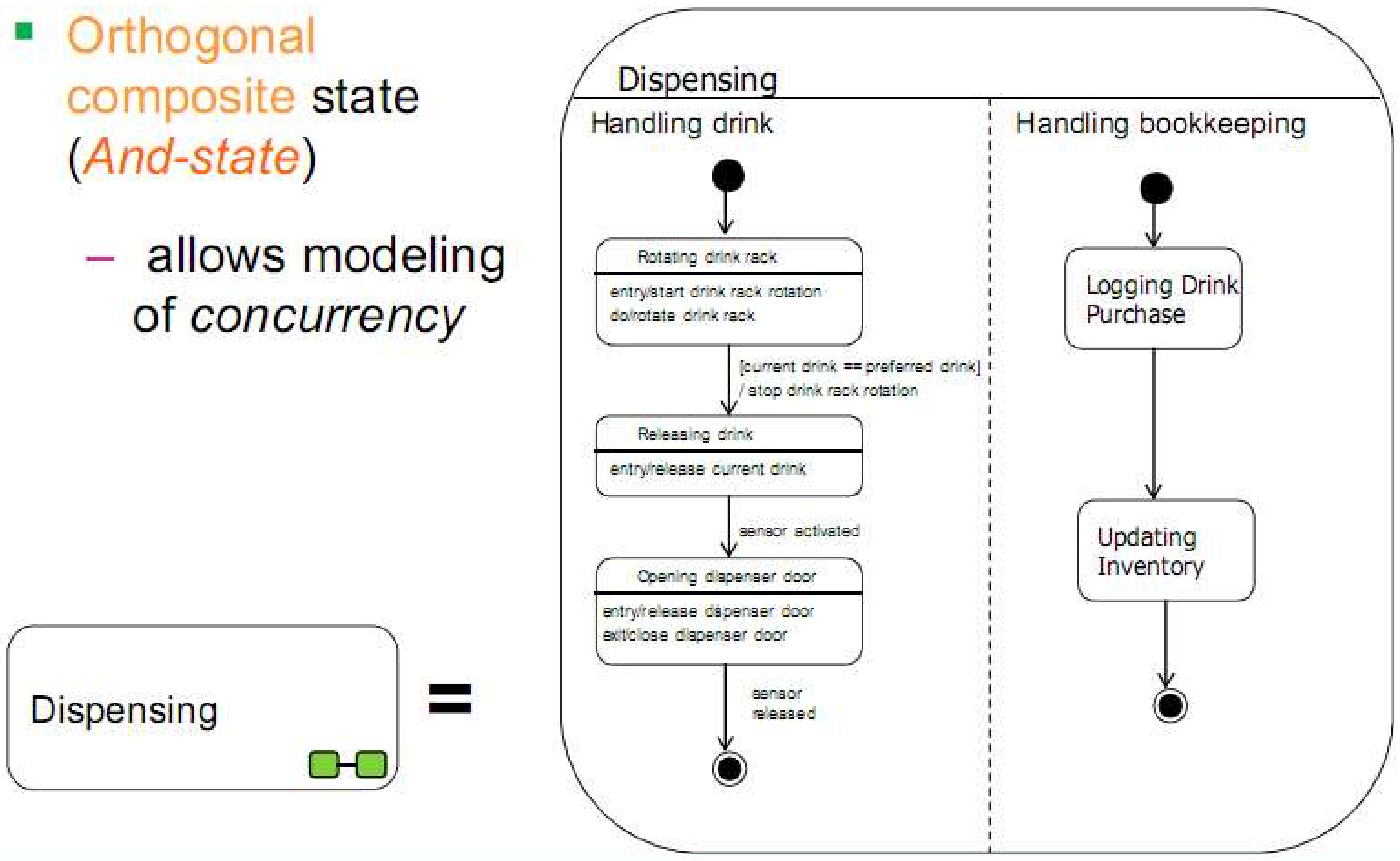


=


Dispensing
Rotating drink rack
entry/start drink rack rotation
do/rotate drink rack
[current drink == preferred drink] / stop drink rack rotation
Releasing drink
entry/release current drink
sensor activated
Opening dispenser door
entry/release dispenser door
exit/close dispenser door
sensor released

State Types: Composite States

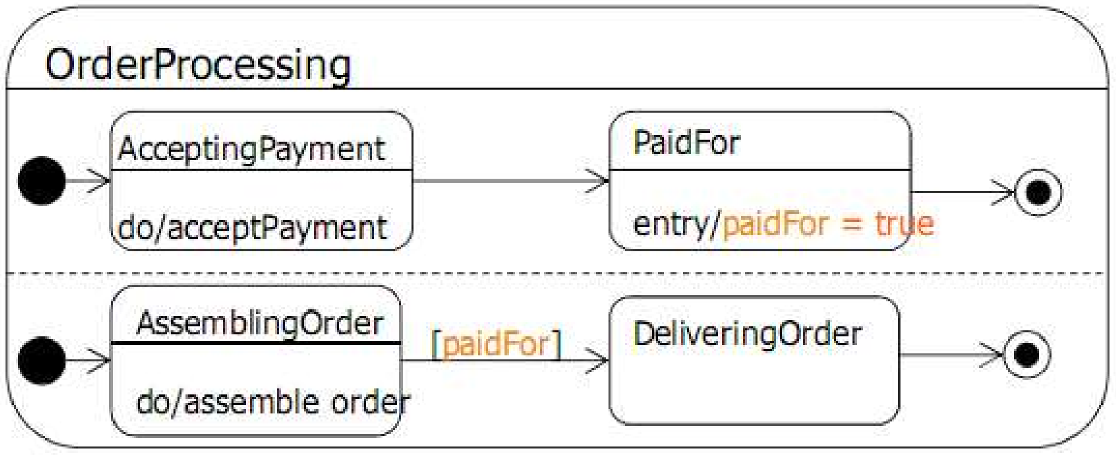
- Orthogonal composite state (*And-state*)
  - allows modeling of *concurrency*



Composite States -- Synchronization

- *Asynchronous communication* is achieved by one substate setting a flag for another one to process in its own time.
  - Use attributes of the context object as flags

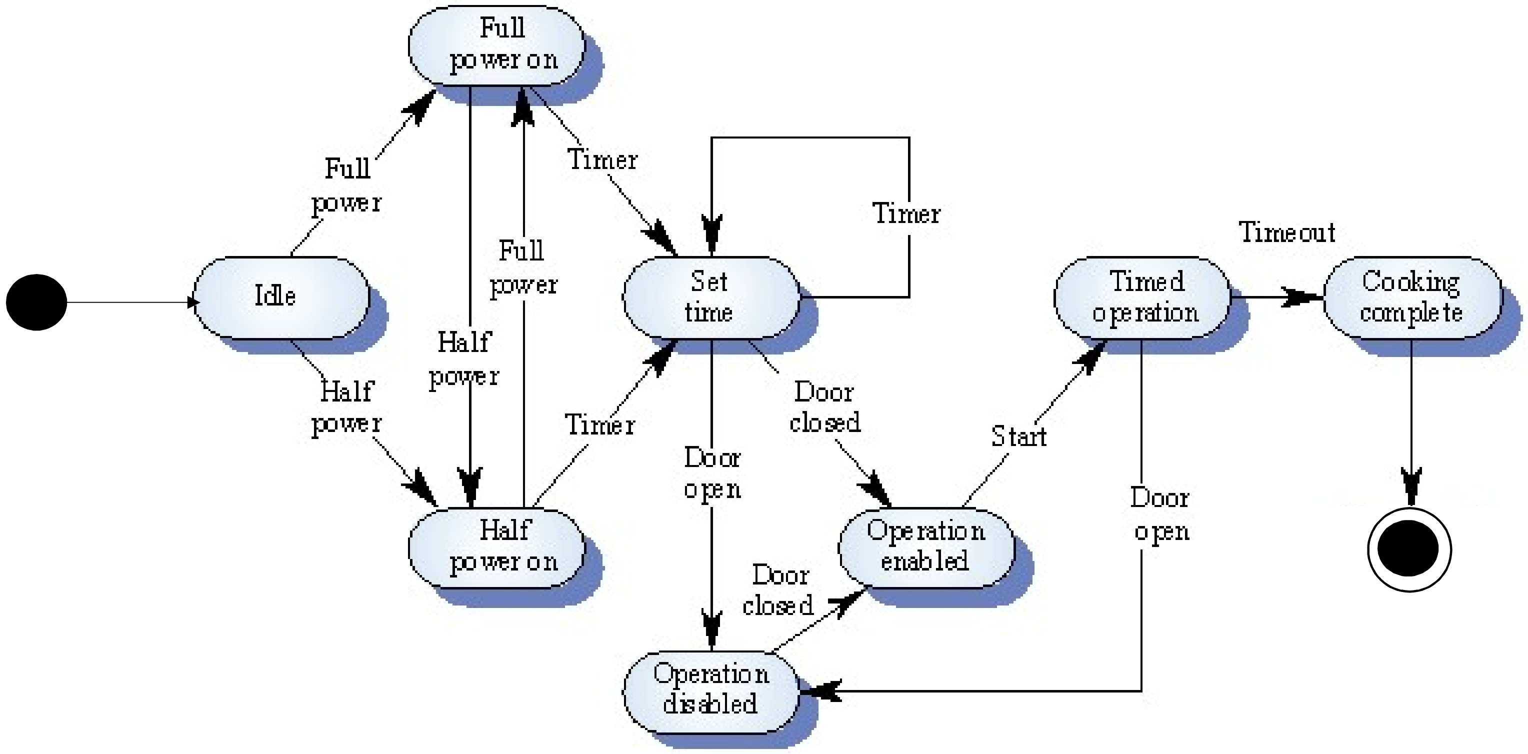
Substate communication using the attribute PaidFor as a flag: The upper substate sets the flag and the lower substate uses it in a guard condition



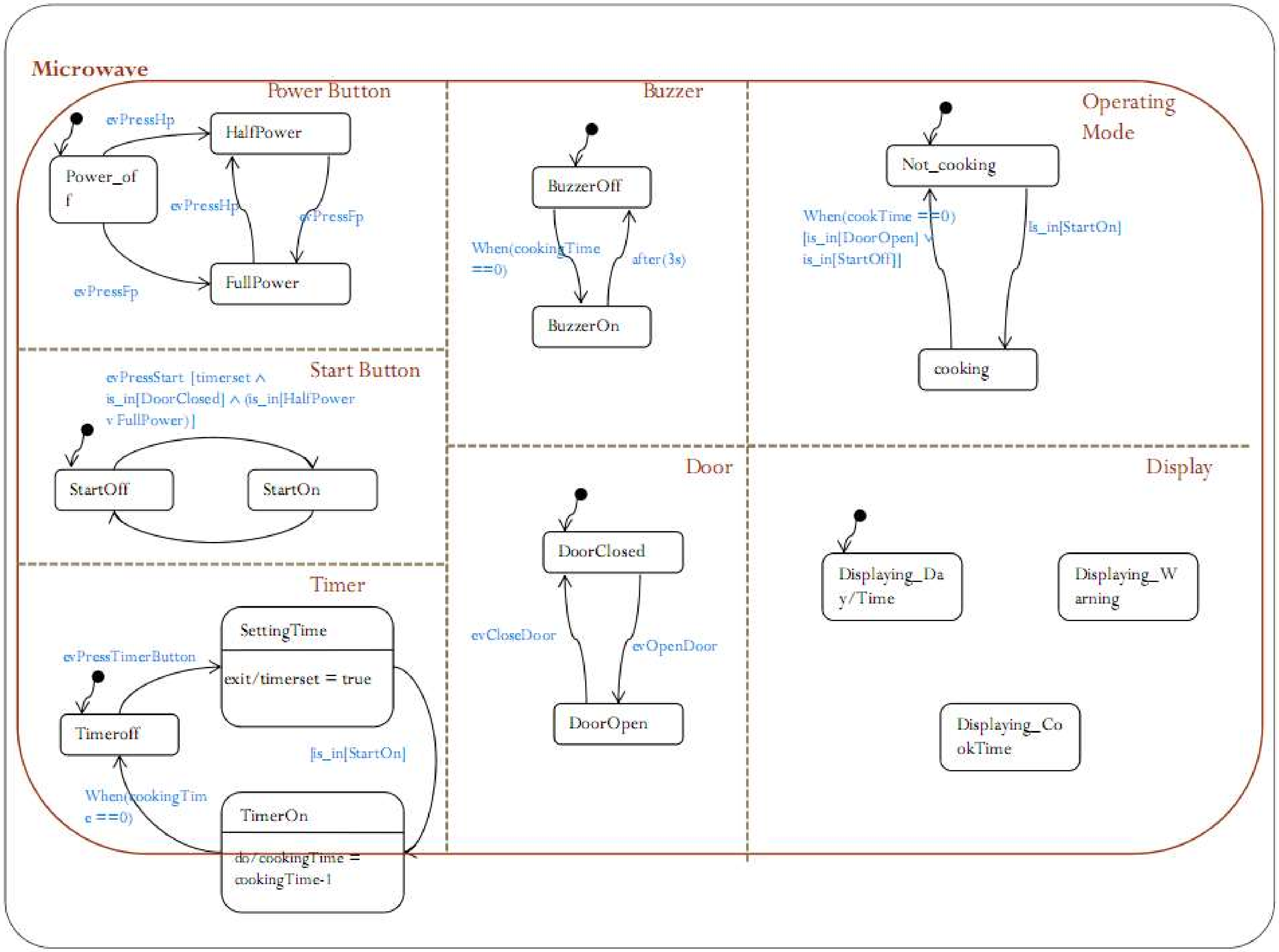
Exercise 1 -- Microwave

- The purpose of this exercise is to develop a *statechart* for a simple microwave oven. The oven is equipped with buttons to set the power and the timer and to start the system. We assume that the sequence of actions in using the microwave is:
  - Select the power level (either half-power or full-power).
  - Input the cooking time.
  - Press start and the food is cooked for the given time.

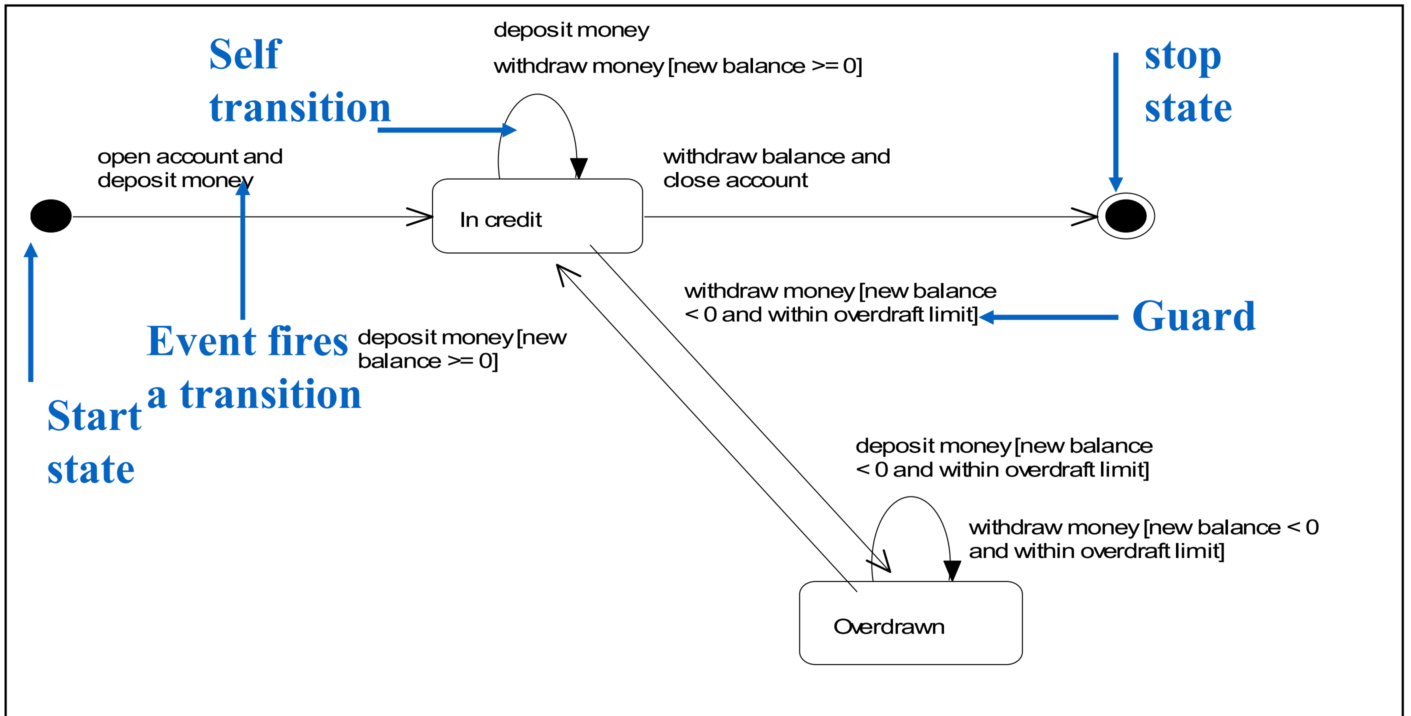
  For safety reasons, the oven should not operate when the door is open and, on completion of cooking, a buzzer is sounded.



Full power on
Full power
Timer
Idle
Half power
Full power
Set time
Timer
Timeout
Timed operation
Cooking complete
Half power
Half power on
Timer
Door open
Door closed
Start
Door open
Operation enabled
Door closed
Operation disabled

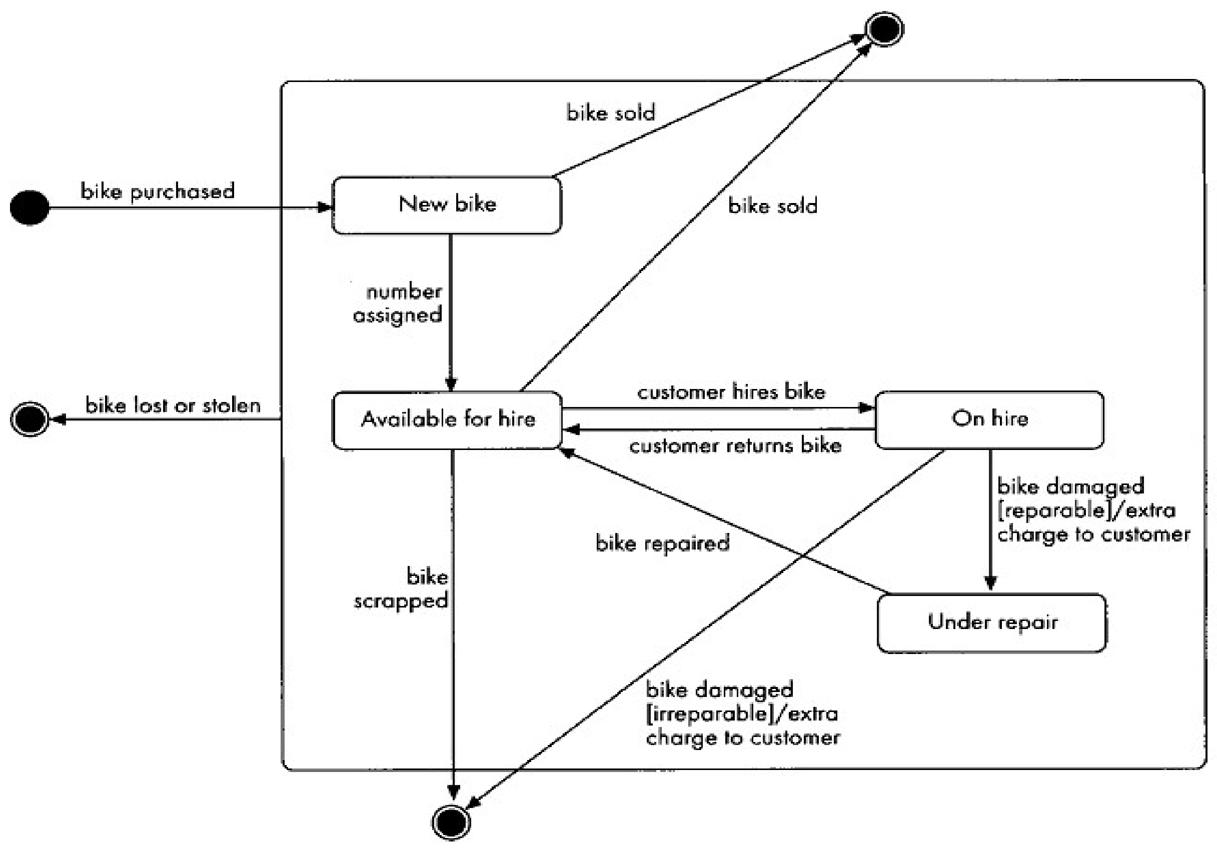

Microwave
Power Button
evPressHp
HalfPower
Power_of f
evPressHp
evPressFp
FullPower
evPressFp
Buzzer
BuzzerOff
When(cookingTime ==0)
after(3s)
BuzzerOn
Operating Mode
Not_cooking
When(cookTime ==0) [is_in[DoorOpen] v is_in[StartOff]]
Is_in[StartOn]
cooking
Start Button
evPressStart [timerset ∧ is_in[DoorClosed] ∧ (is_in[HalfPower v FullPower)]
StartOff
StartOn
Timer
SettingTime
exit/timerset = true
evPressTimerButton
Timeroff
[is_in[StartOn]
When(cookingTime ==0)
TimerOn
do/cookingTime = cookingTime-1
Door
DoorClosed
evCloseDoor
evOpenDoor
DoorOpen
Display
Displaying_Day/Time
Displaying_Warning
Displaying_CookTime

State diagram for a bank account class

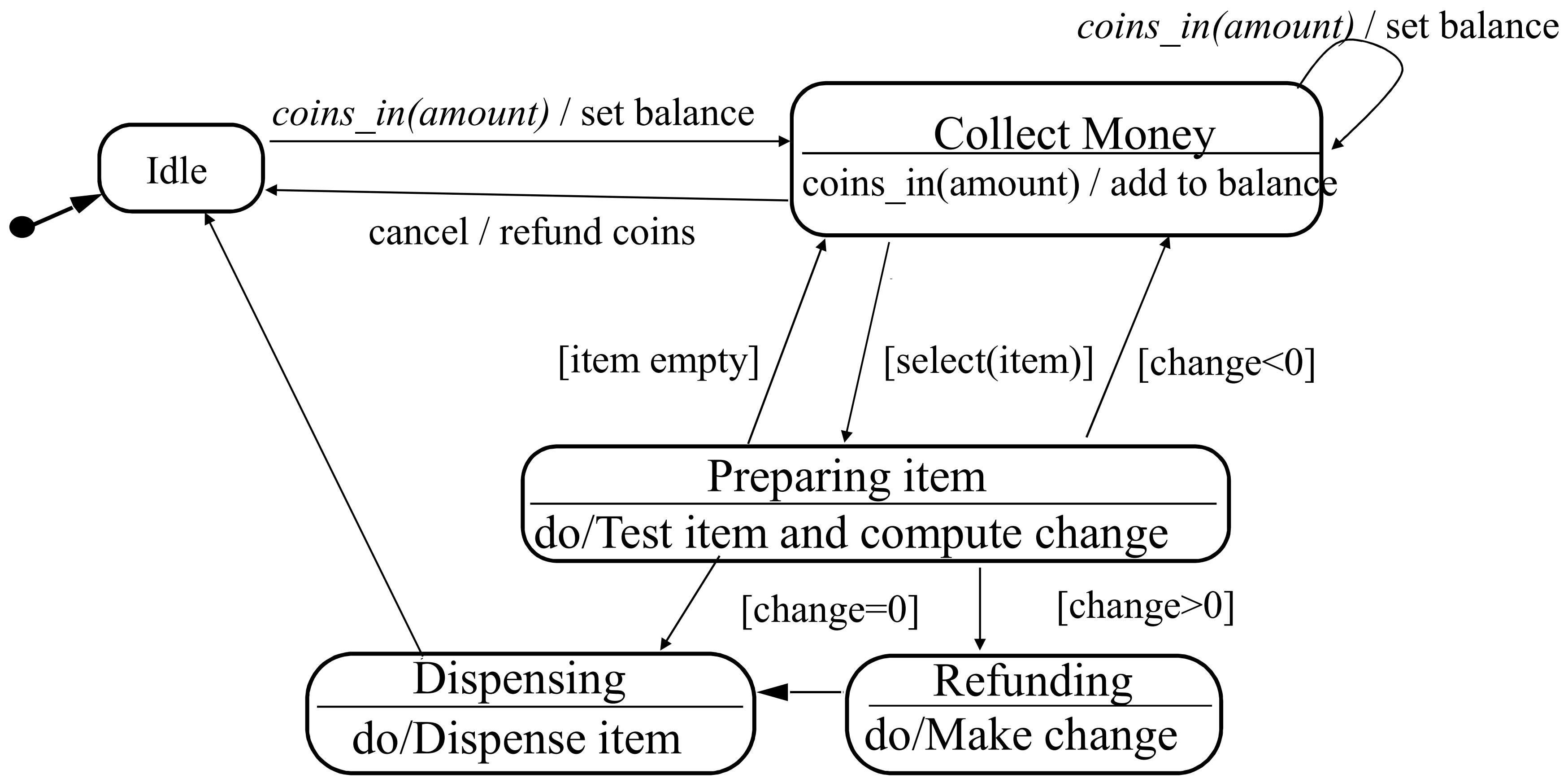

deposit money
withdraw money [new balance >= 0]
Self transition
stop state
open account and deposit money
withdraw balance and close account
In credit
withdraw money [new balance < 0 and within overdraft limit]
Guard
Event fires a transition
deposit money [new balance >= 0]
Start state
deposit money [new balance < 0 and within overdraft limit]
withdraw money [new balance < 0 and within overdraft limit]
Overdrawn

state diagram for a bank account class

Completed state diagram for Bike Object



Example of a State Chart Diagram(Vending Machine)



Example: Job Application

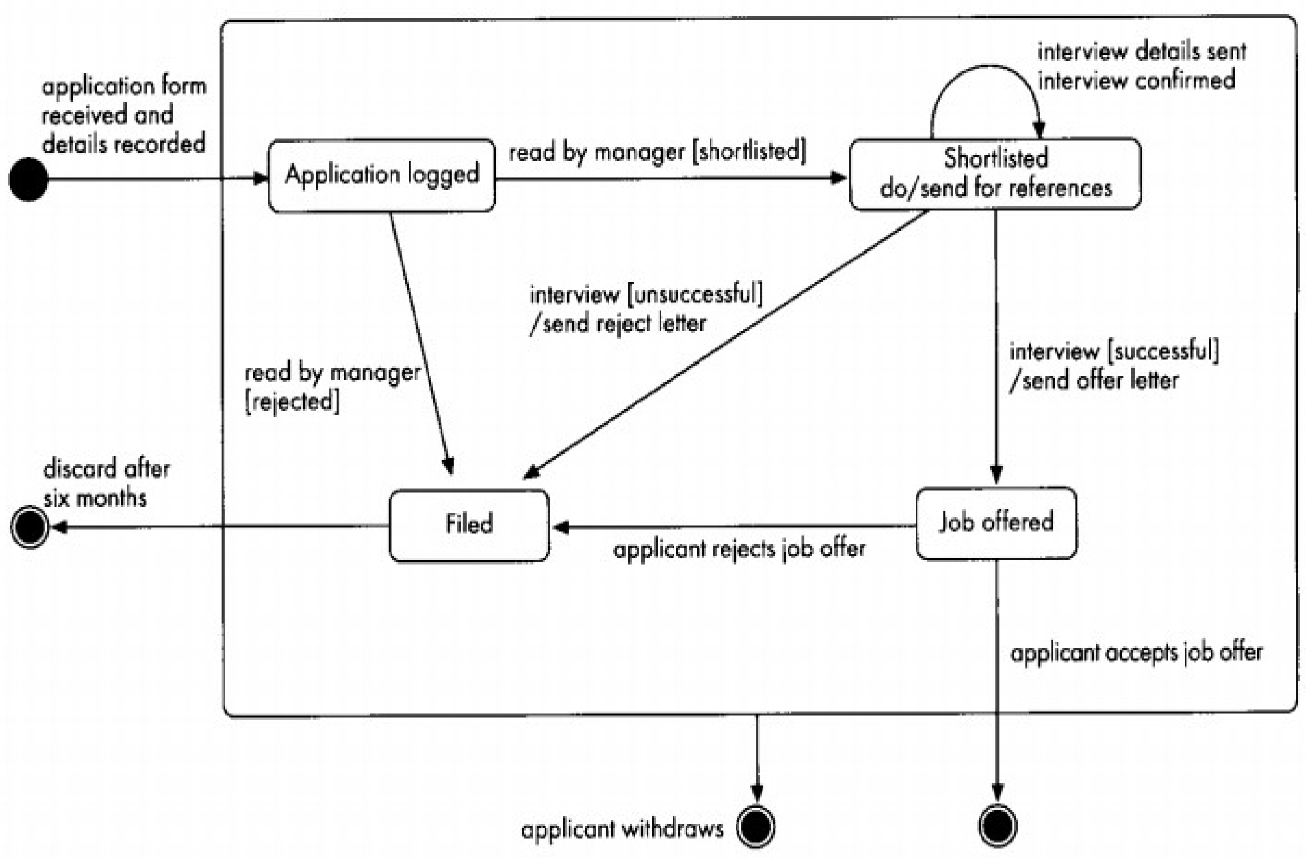

application form received and details recorded
Application logged
read by manager [shortlisted]
Shortlisted do/send for references
interview details sent
interview confirmed
interview [unsuccessful] /send reject letter
read by manager [rejected]
interview [successful] /send offer letter
discard after six months
Filed
applicant rejects job offer
Job offered
applicant accepts job offer
applicant withdraws

Figure 7.7 Completed state diagram with superstate for the Job Application class

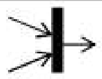
Outline

- Statechart Diagrams
  - Behavioral State Machines
    - States
    - Transitions
    - Composite States
    - Pseudostates
    - Submachine States

Pseudostates

- Pseudostates are not in any way states, but they are used to represent specific behavior during transitions between regular states
- Can represent complex state changes within a state machine

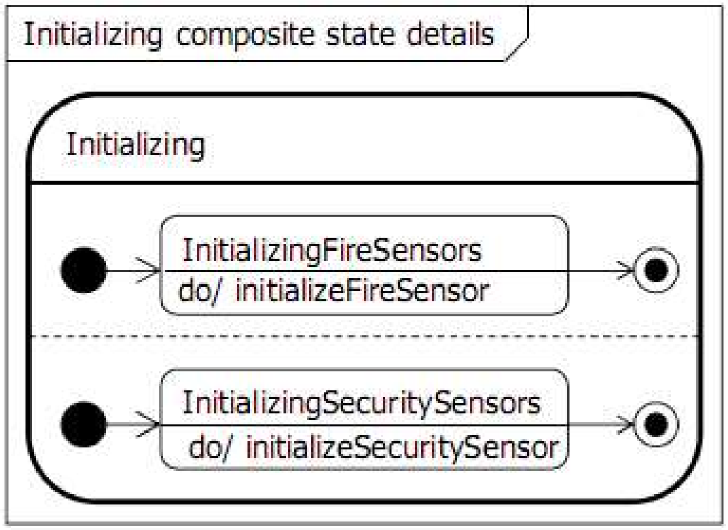
Pseudostates: Types

Pseudostate name	Symbol	Description
Initial pseudostate	●	The starting point of a state machine.
Choice	◇	Allows the execution of a state machine to choose between several different states based on gard conditions.
Deep History	Ⓗ*	Indicates the state machine should resume the last substate it was in within a region no matter "how deep" the substate is.
Entry point	○	Represents a possible target for a transition into a composite state. Must be labeled with a name.
Exit point	⊗	Represents a possible source for transition from a composite state. Must be labeled with a name.
Fork and join		Represents a split in the execution of a state machine. The join reunites the regions into a single transition. The state machine won't transition from the join until all regions have transitioned to the join pseudostate.
Junction	●	Brings several possible transitions together into one pseudo state. One or more transitions may leave the junction
Shallow history	Ⓗ	Indicates the state machine should resume the last substate it was in within a region. Substate must be at the same level as history pseudostate
Terminate node	×	Causes the state machine to terminate

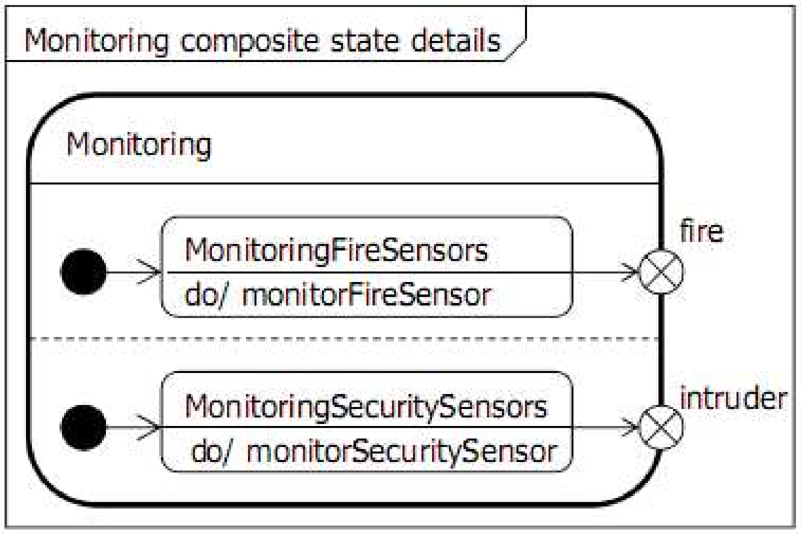
Implicit Fork

- When we enter the superstate, both submachines start executing concurrently - this is an implicit fork

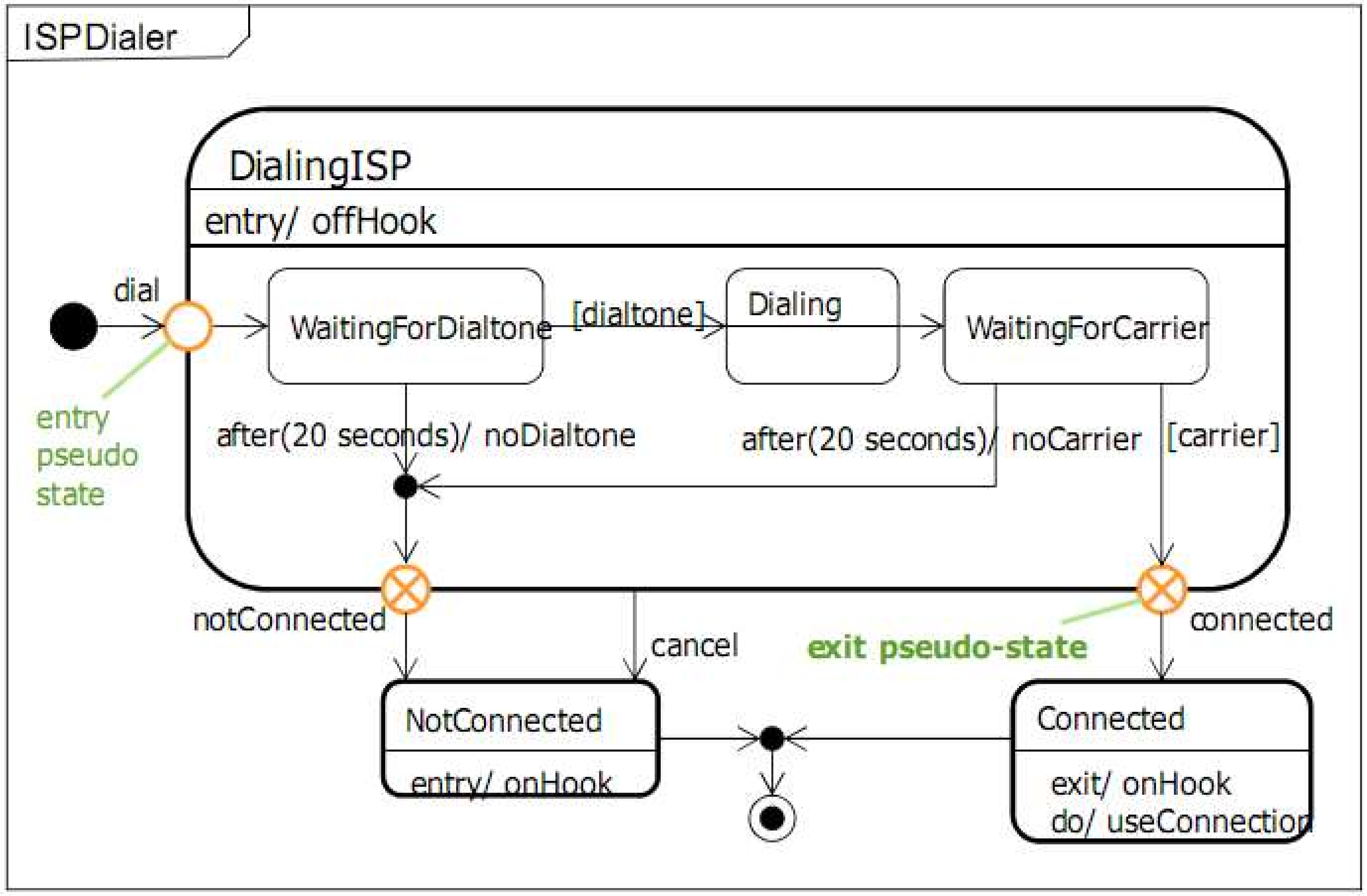
Synchronized exit - exit the superstate when both regions have terminated


Initializing composite state details
Initializing
InitializingFireSensors
do/ initializeFireSensor
InitializingSecuritySensors
do/ initializeSecuritySensor

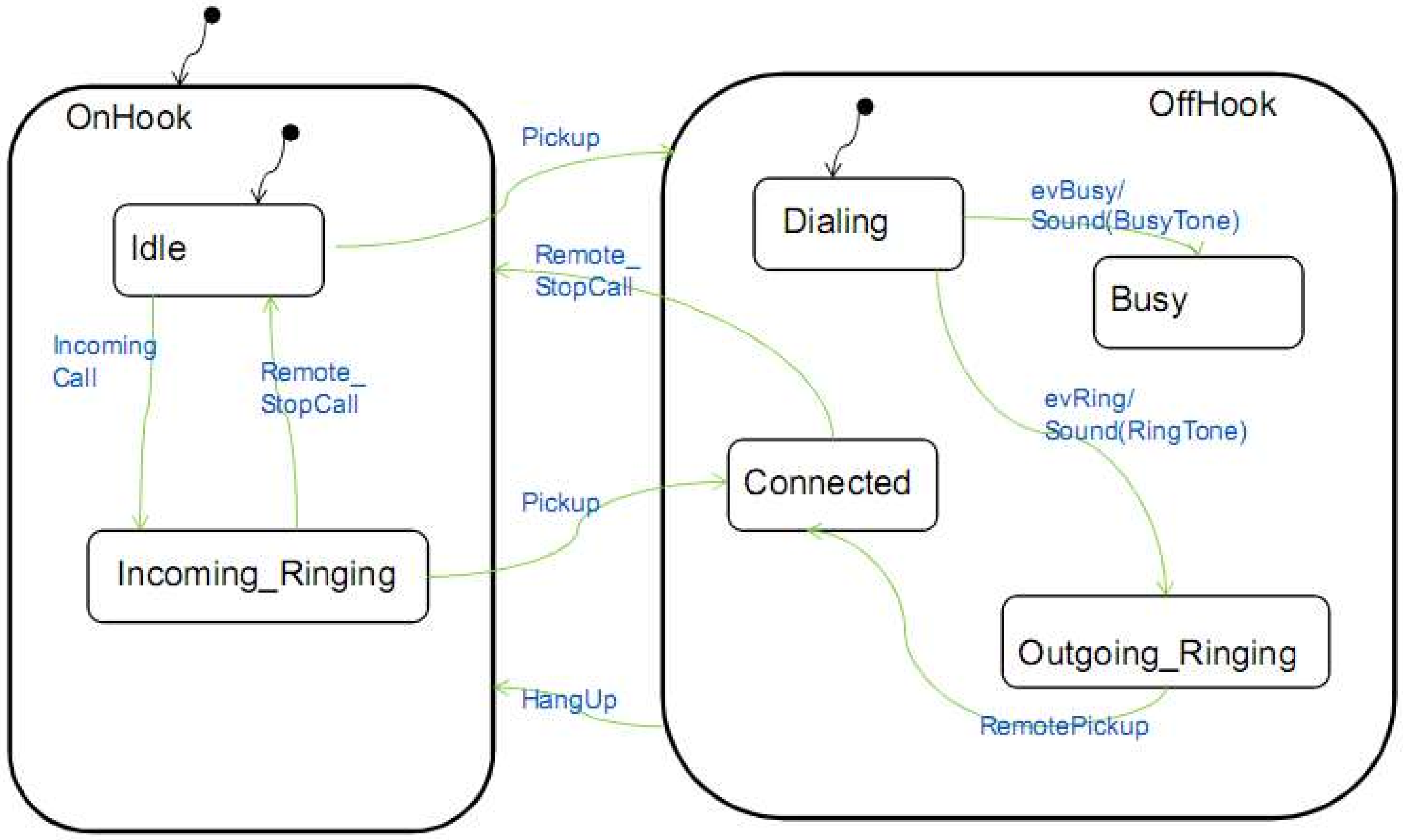
Unsynchronized exit - exit the superstate when either region terminates. The other region continues


Monitoring composite state details
Monitoring
MonitoringFireSensors
do/ monitorFireSensor
fire
MonitoringSecuritySensors
do/ monitorSecuritySensor
intruder

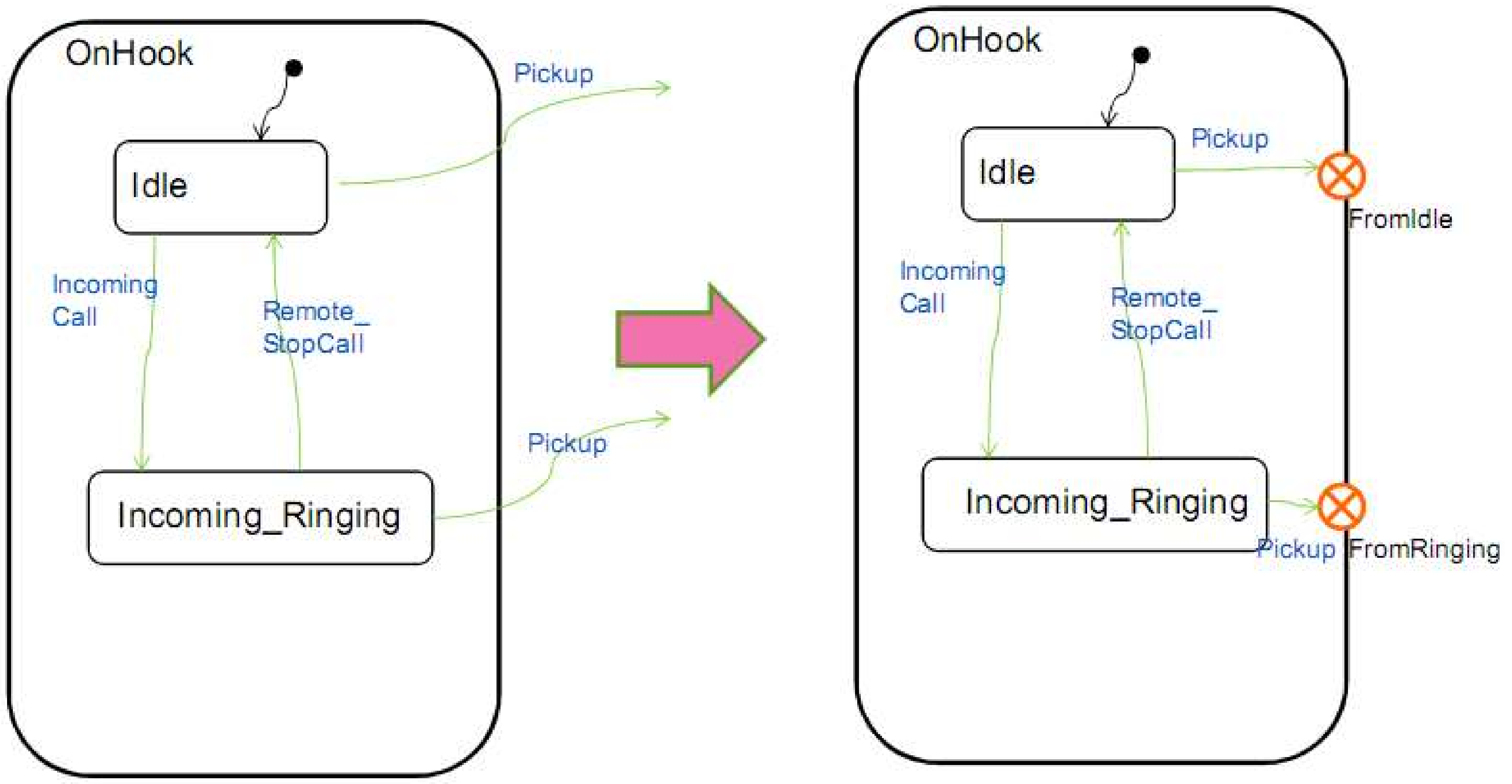
Entry and Exit Point



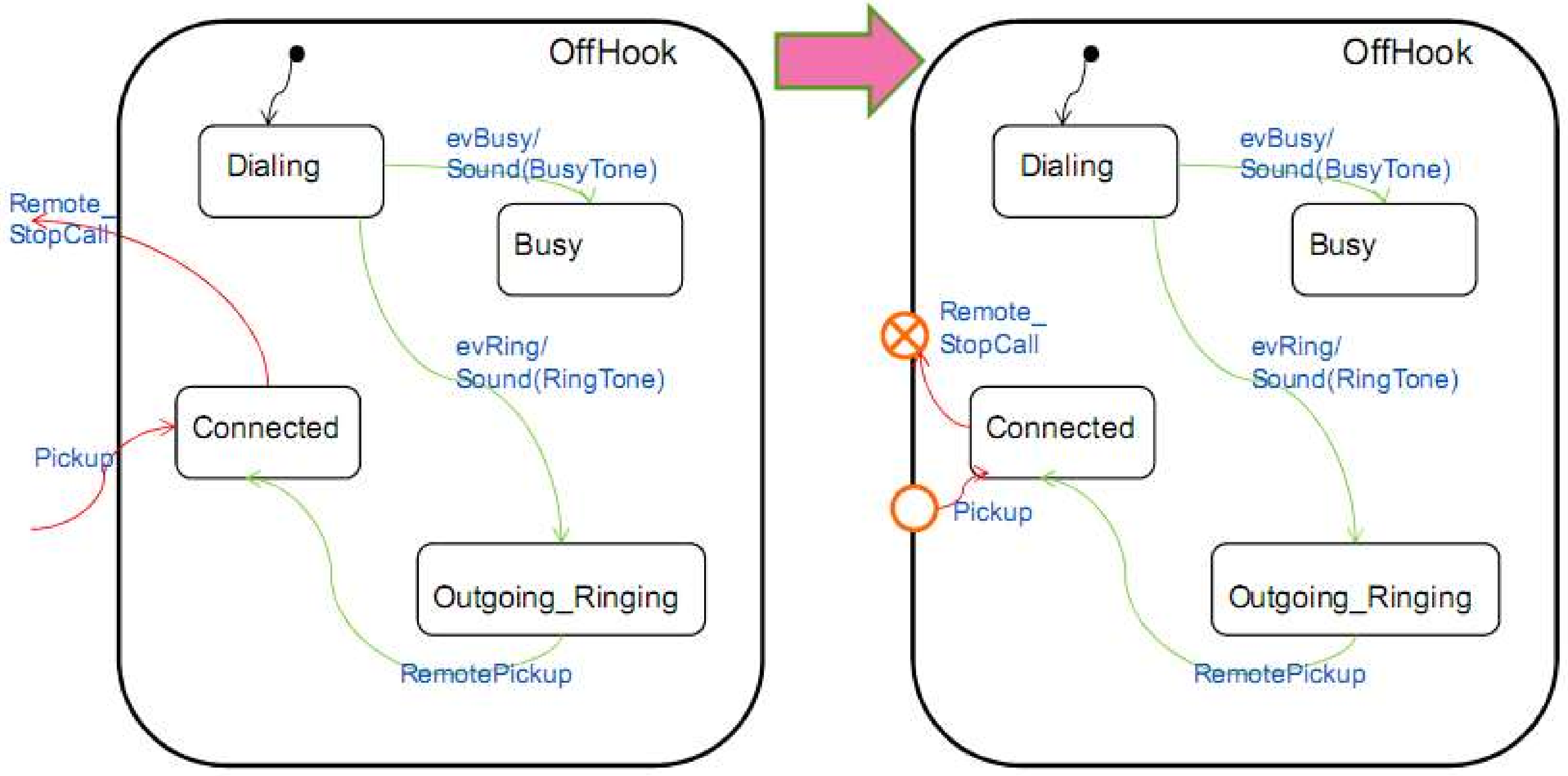
Submachines – Telephone Example

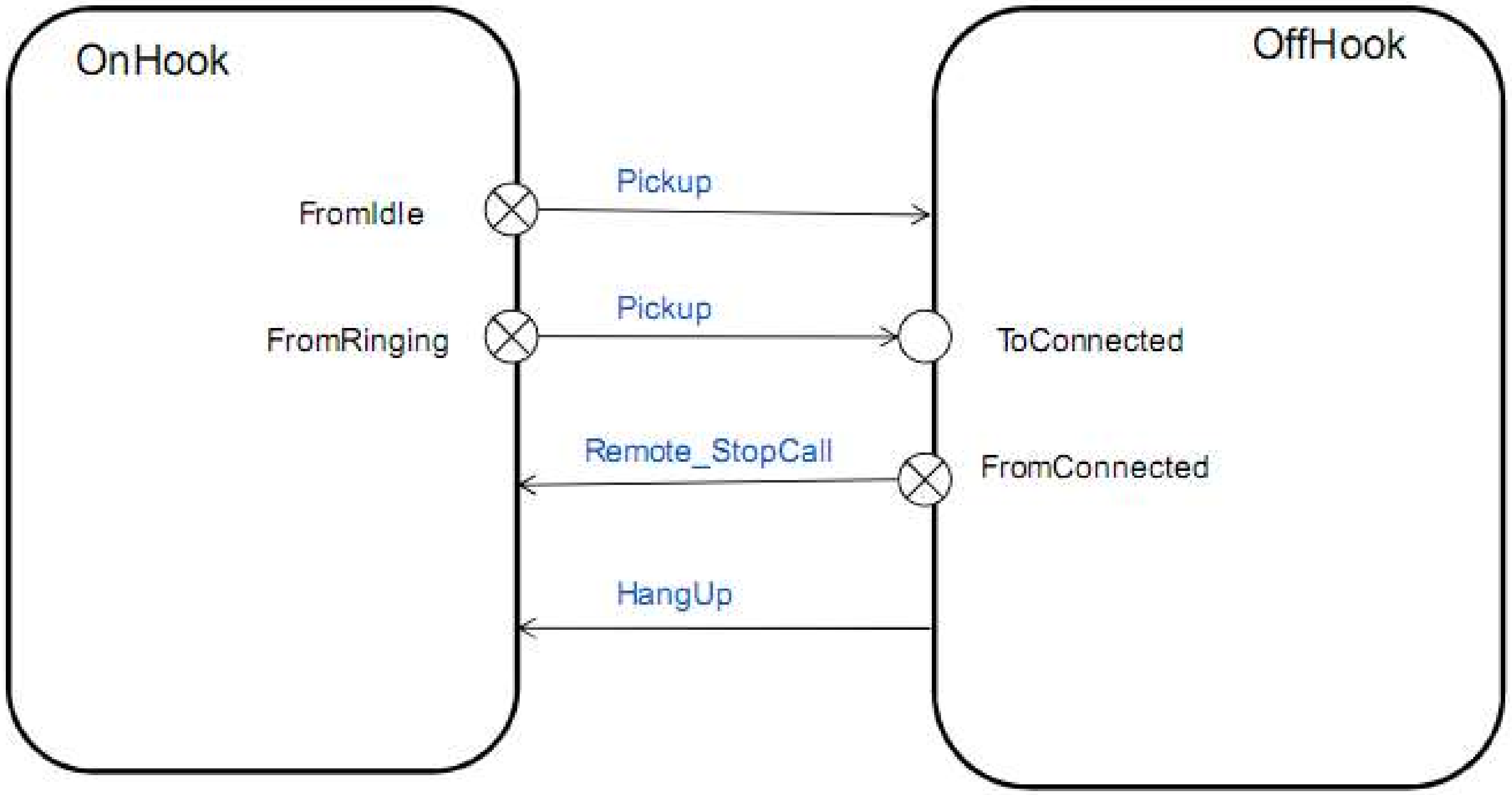


Submachines – Telephone Example



Submachines – Telephone Example




OnHook
OffHook
FromIdle
Pickup
FromRinging
Pickup
ToConnected
Remote_StopCall
FromConnected
HangUp