Software Architecture Theory

E02. *UML 2.0 State Machine Diagram*

2014

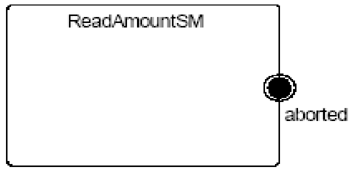
Sungwon Kang

Table of Contents

1. Basics
2. How to Model a State Machine Diagram
3. Advanced Topics

1. Basics

Introduction



- An object can respond to a wide range of messages throughout its lifetime.
- In UML, *state machines* are used for specifying responses to the events detected during the lifetime of an object.
- *States* is an important way of organizing "behavior" of an object.
  - Not all possible and/or legal states of an object are shown by *interaction diagrams*.
- Interaction diagrams and state machines are two complementary views of a system.

Why?

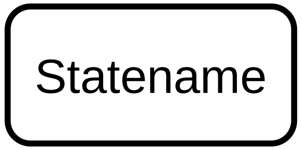
State Machine

- **Behavioral state machine**
  - Specify behavior of various model elements (e.g., class instances).
  - Object based variant of Harel statecharts (hierarchical modeling of a state machine)

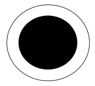
- **Protocol state machine**
  - Used to express usage protocols:
    - Define a lifecycle for objects, or an order of the invocation of its operation.
    - Legal usage scenarios of classifiers, interfaces, and ports can be associated

States

- An object has a number of possible states and is in exactly one of these states at any given time
- An object can change its *current state* and its state at any given time depends on its history.
- Depending on its state, an object may respond differently to the same stimulus
- Stimuli (= events) cause the state to transition from one state to another.
- When the *final state* is entered, its containing region is completed



State



Final state

Example

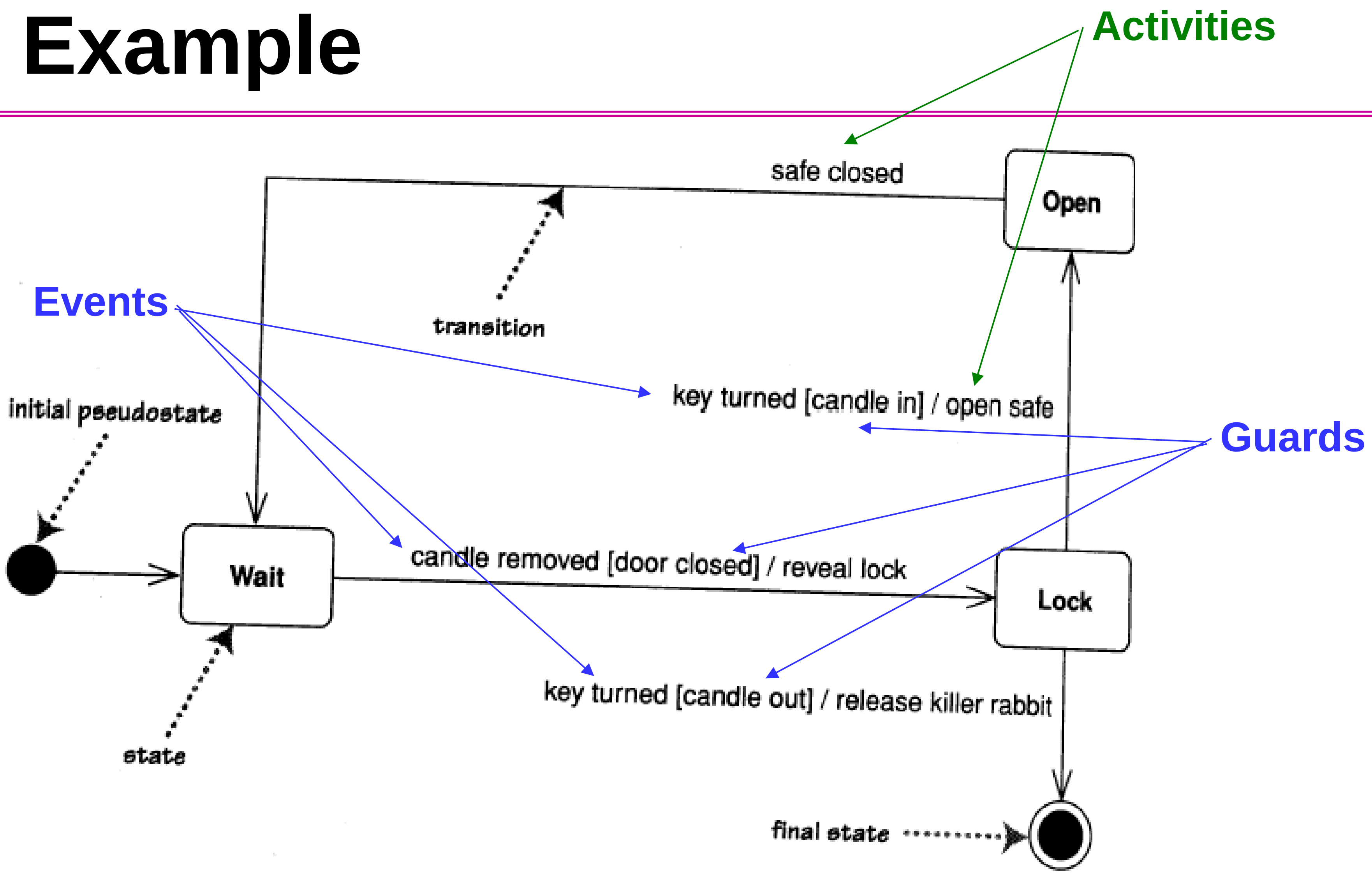

Activities
safe closed
Open
transition
Events
initial pseudostate
key turned [candle in] / open safe
Guards
Wait
candle removed [door closed] / reveal lock
Lock
key turned [candle out] / release killer rabbit
state
final state

Figure 10.1 *A simple state machine diagram* **Source: [Fowler 04]**

Internal Activities

- States can react to events without transitions.
- Reserved internal activity labels:

  (1) entry: occurs when the state has been entered

  (2) exit: occurs when the state is being exited

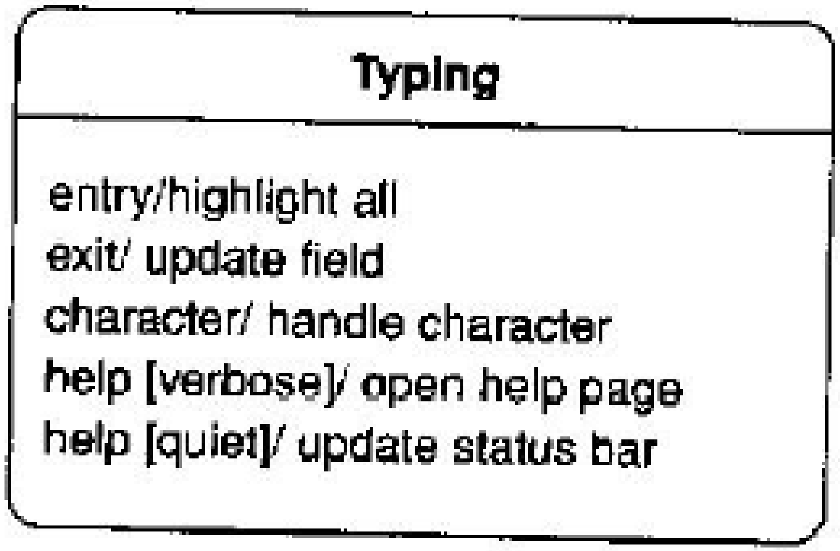
  (3) do: occurs while the state is being occupied

- Syntax
  - label / activity

Example

- entry / numberOfStudents = 0 -- initialization
- exit / Obj -> include(this) -- ensure that Obj include this object
- do / refreshStudentList

Internal Activities


Typing
entry/highlight all
exit/ update field
character/ handle character
help [verbose]/ open help page
help [quiet]/ update status bar

[Fowler 04]

Figure 10.2 *Internal events shown with the typing state of a text field*

Activity State

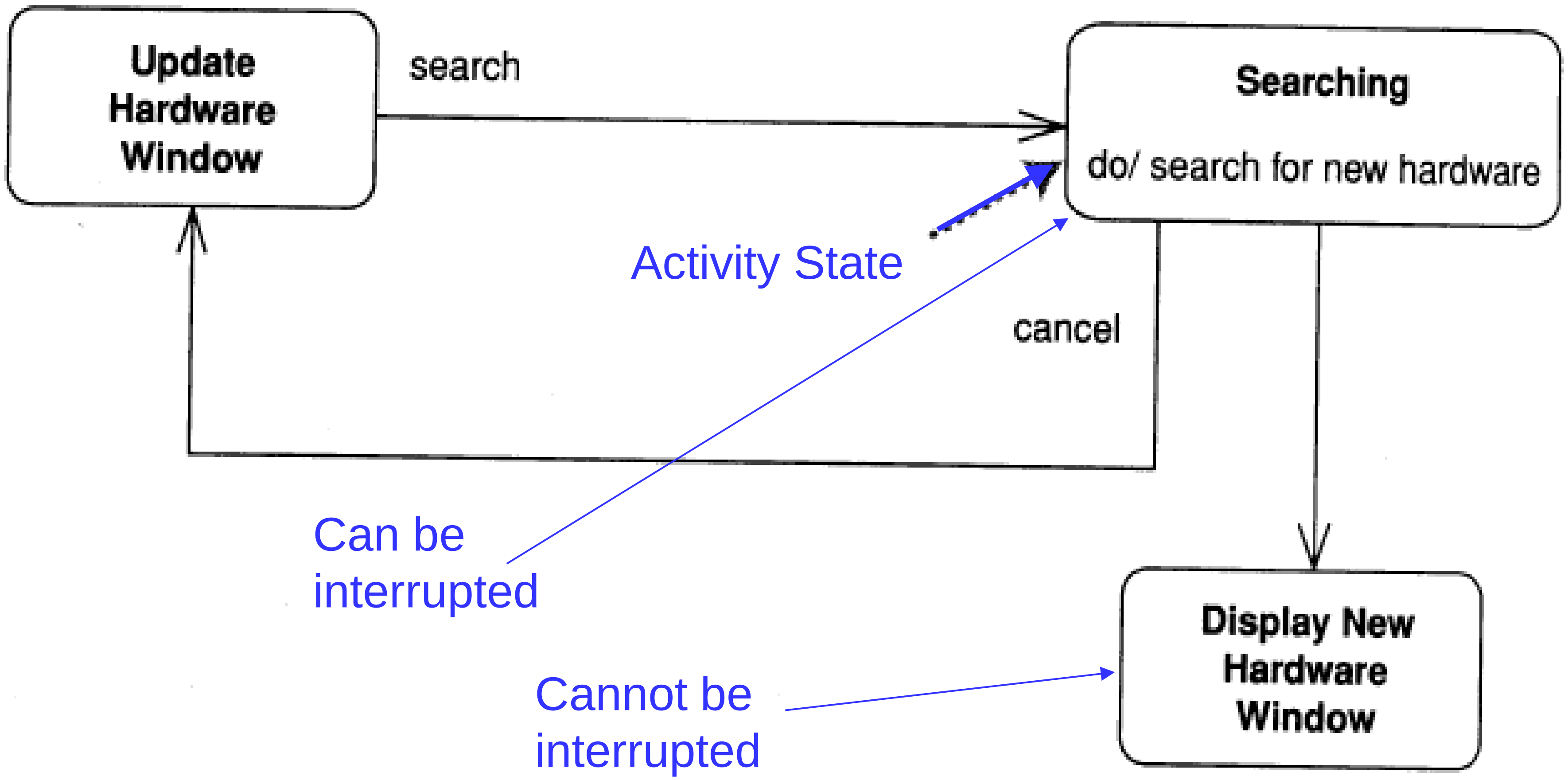
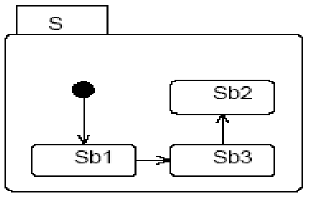


Figure 10.3 *A state with an activity* [Fowler 04]

Substates and Composite States



- There are two kinds of states
  - Simple state: a state with no substructure
  - Composite state: a state comprised of either concurrent (= orthogonal) substates or sequential (= disjoint) substates.
- A substate (= nested state) is a state that is part of a composite state.
- A concurrent substate is an othogonal substate that can be held simultaneously with other substates contained in the same composite state
- A sequential substate is a substate that cannot exist simultaneously with other substates within a composite state.

=> Given two or more substates at the same level, either an object is in one sequential substate or in each of all the concurrent substates.

Composite State

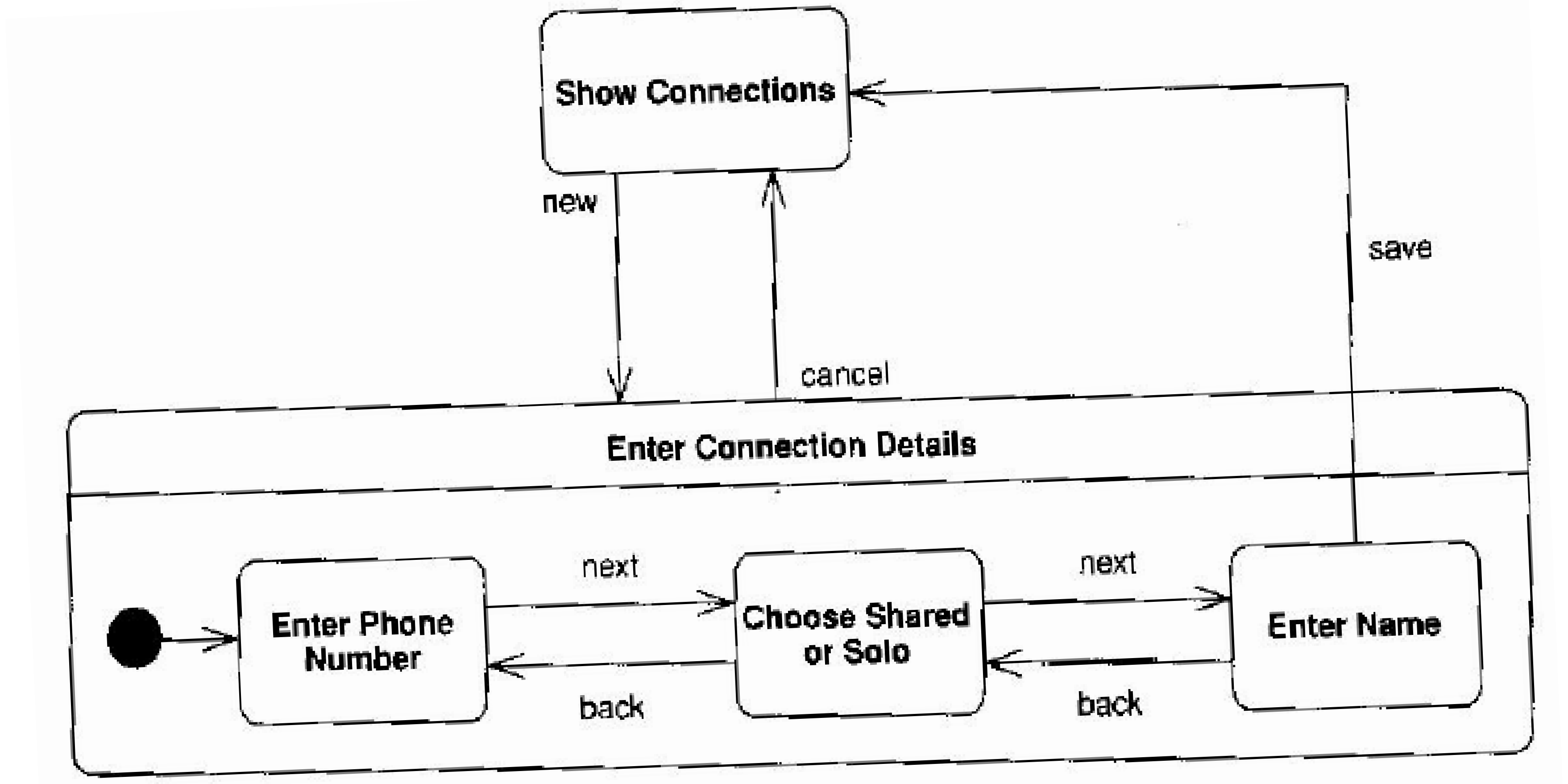
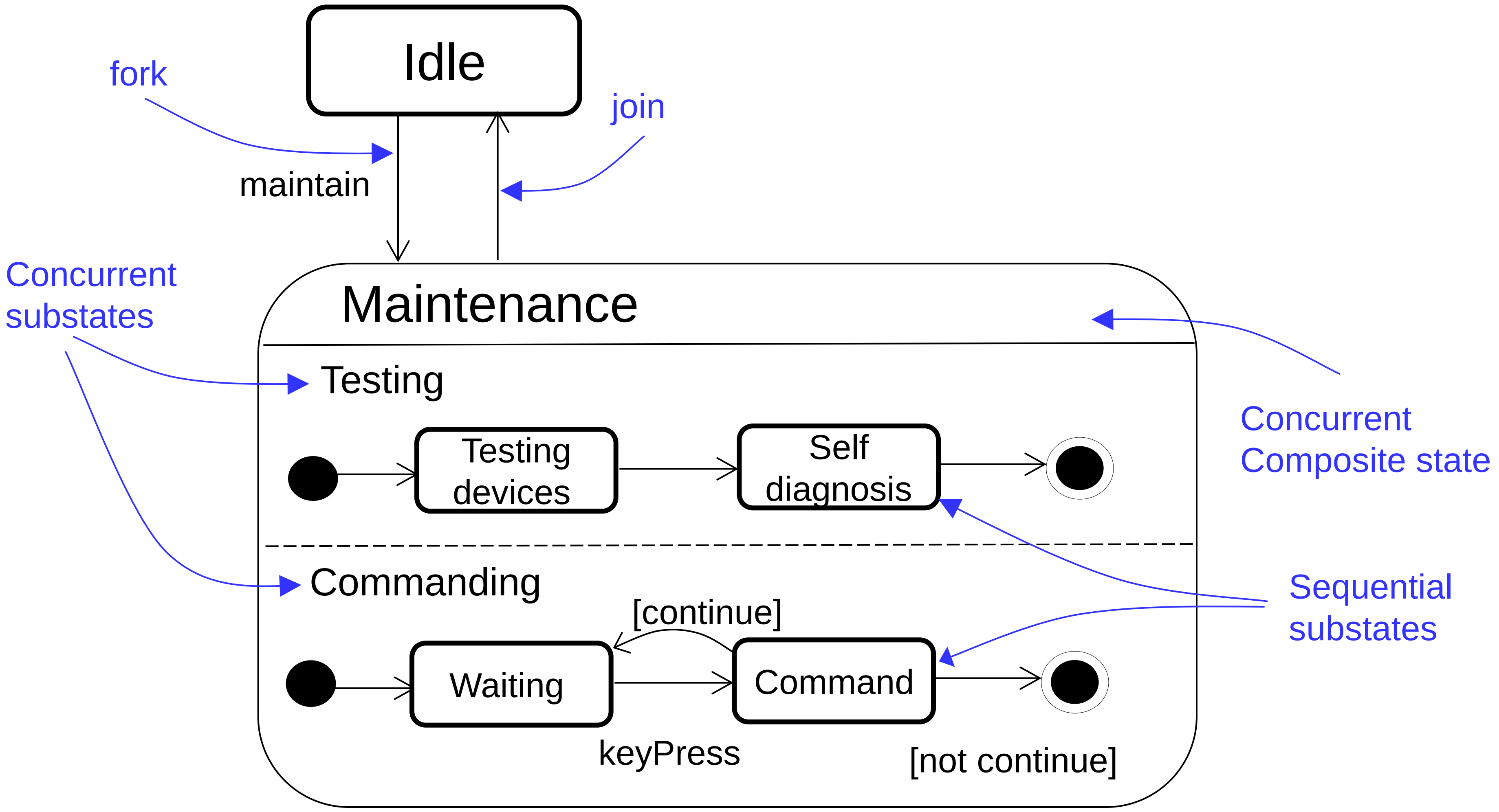


Figure 10.4 *Superstate with nested substates*

Source: [Fowler 04]

Concurrent Substates



Concurrent Substates - Fork and Join

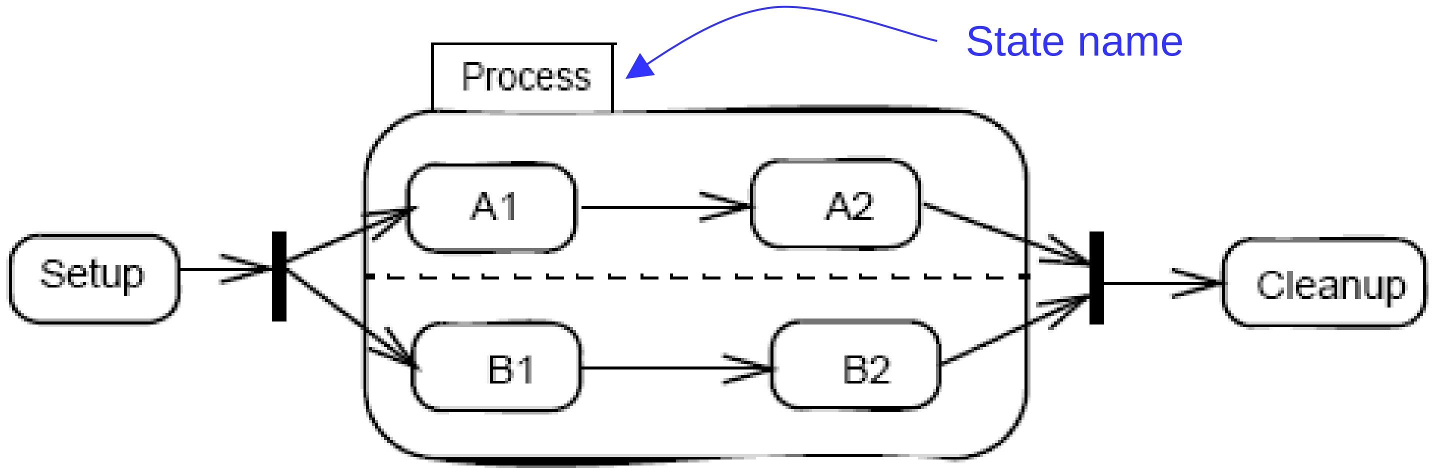


Figure 15.25 - Fork and Join

Concurrent Substates

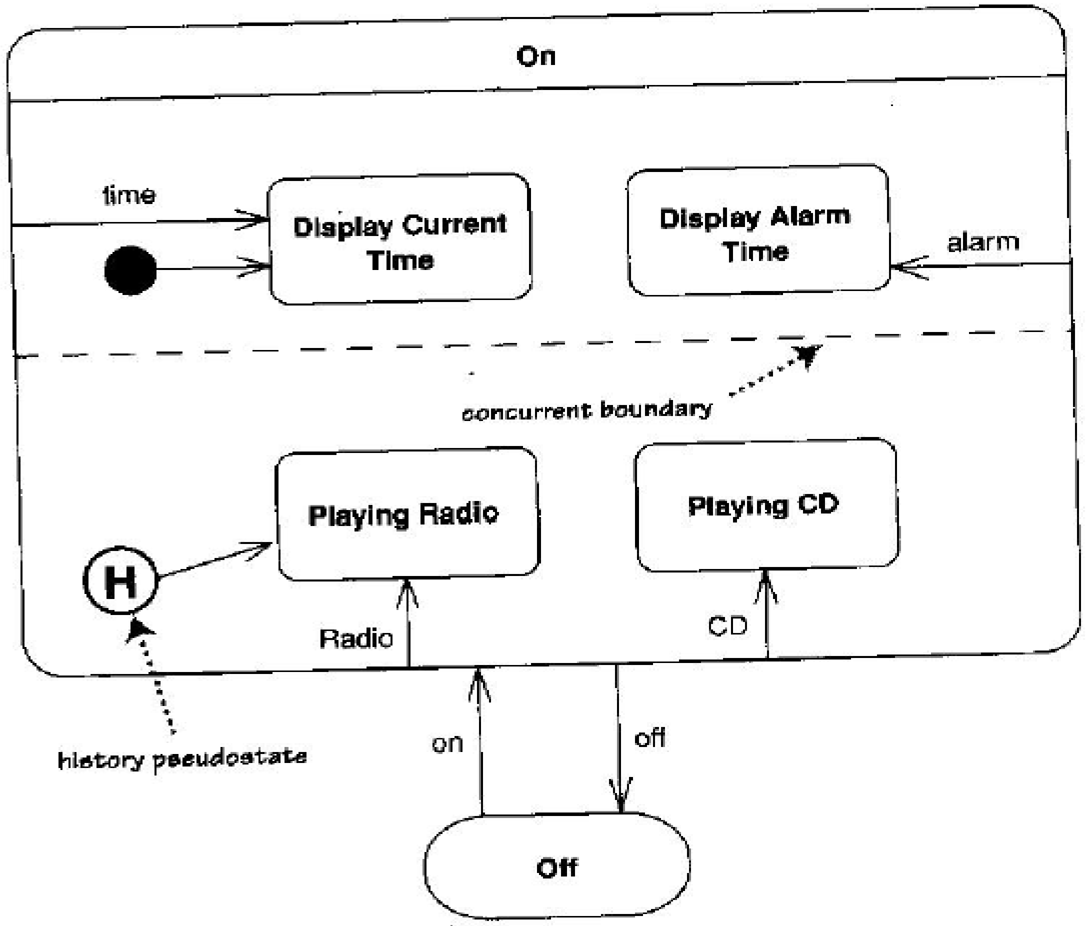


Figure 10.5 Concurrent orthogonal states

Source: [Fowler 04]

There are two states

On is a composite state

When On is entered, two threads execute in parallel

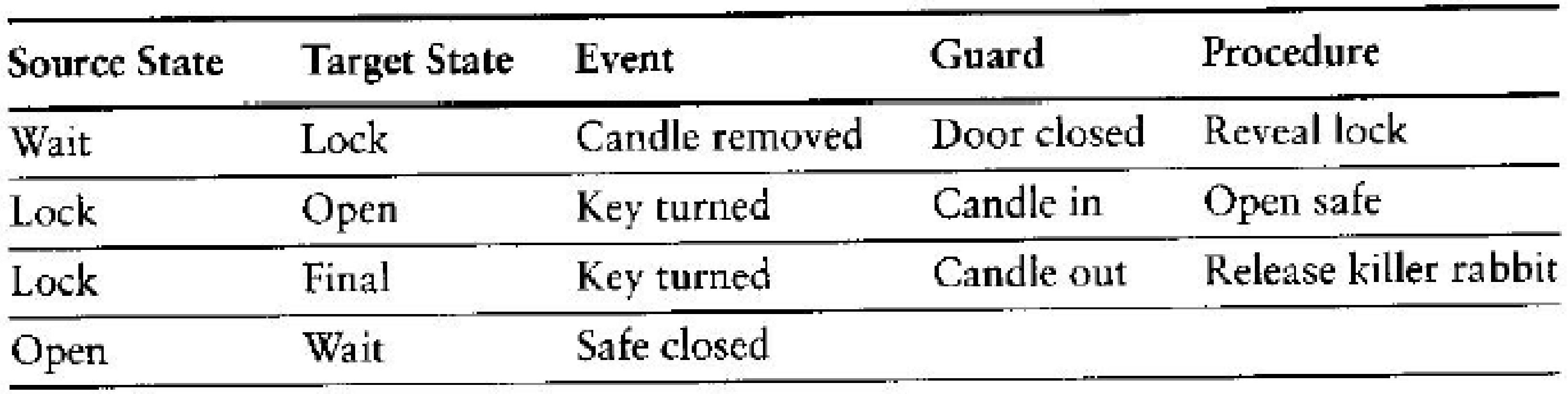
Unless there are any events, it will start with and stay at "Display Current Time"

Unless CD event occurs, the object will start playing radio that it used to play.

Implementing State Diagrams

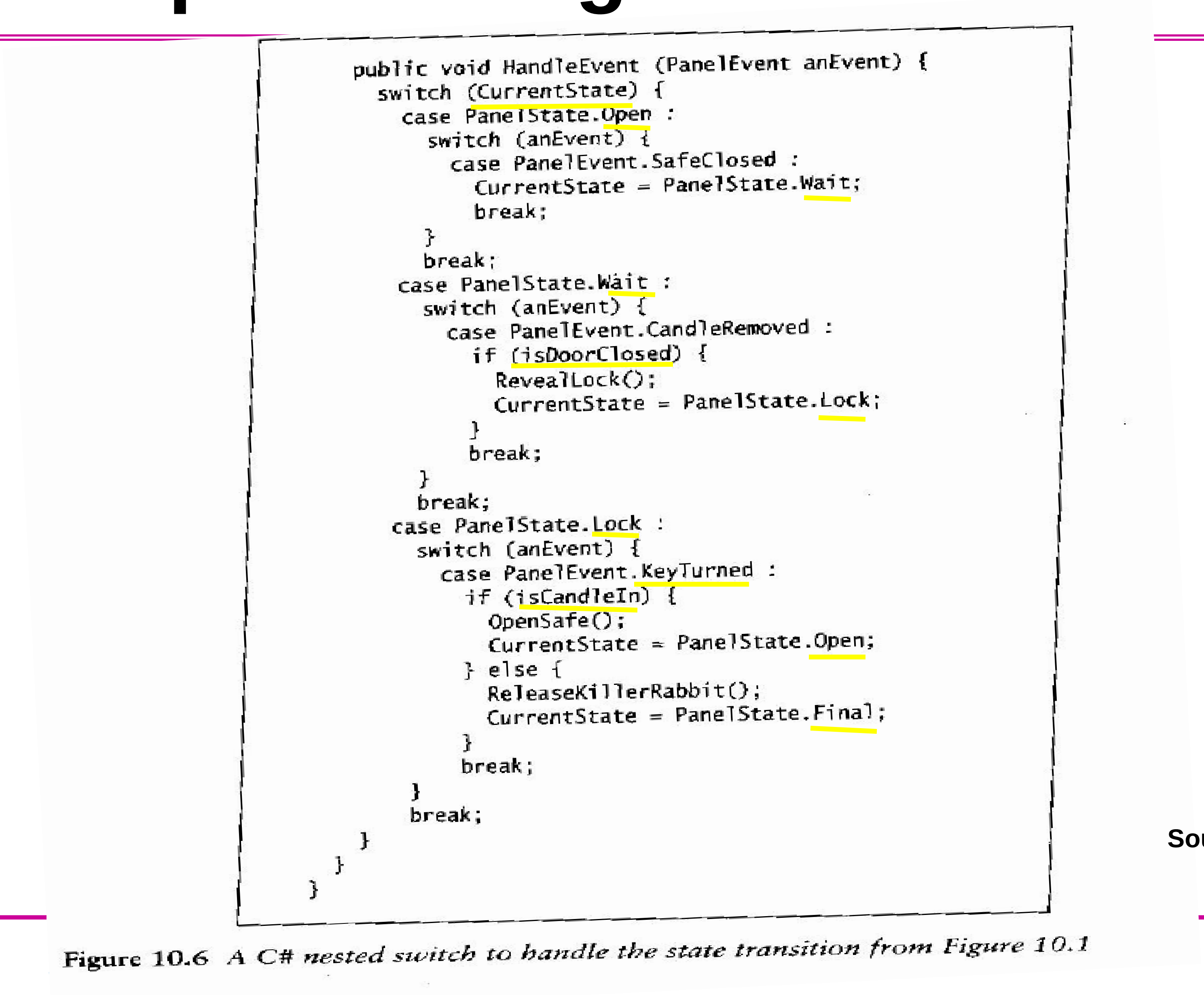
Table 10.1 *A State Table for Figure 10.1*

Source: [Fowler 04]



Source State	Target State	Event	Guard	Procedure
Wait	Lock	Candle removed	Door closed	Reveal lock
Lock	Open	Key turned	Candle in	Open safe
Lock	Final	Key turned	Candle out	Release killer rabbit
Open	Wait	Safe closed		

Implementing State Diagrams



```
public void HandleEvent (PanelEvent anEvent) {
  switch (CurrentState) {
    case PanelState.Open :
      switch (anEvent) {
        case PanelEvent.SafeClosed :
          CurrentState = PanelState.Wait;
          break;
      }
      break;
    case PanelState.Wait :
      switch (anEvent) {
        case PanelEvent.CandleRemoved :
          if (isDoorClosed) {
            RevealLock();
            CurrentState = PanelState.Lock;
          }
          break;
      }
      break;
    case PanelState.Lock :
      switch (anEvent) {
        case PanelEvent.KeyTurned :
          if (isCandleIn) {
            OpenSafe();
            CurrentState = PanelState.Open;
          } else {
            ReleaseKillerRabbit();
            CurrentState = PanelState.Final;
          }
          break;
      }
      break;
  }
}
}
```

Figure 10.6 *A C# nested switch to handle the state transition from Figure 10.1*

Source: [Fowler 04]

2. How to Model a State Machine Diagram

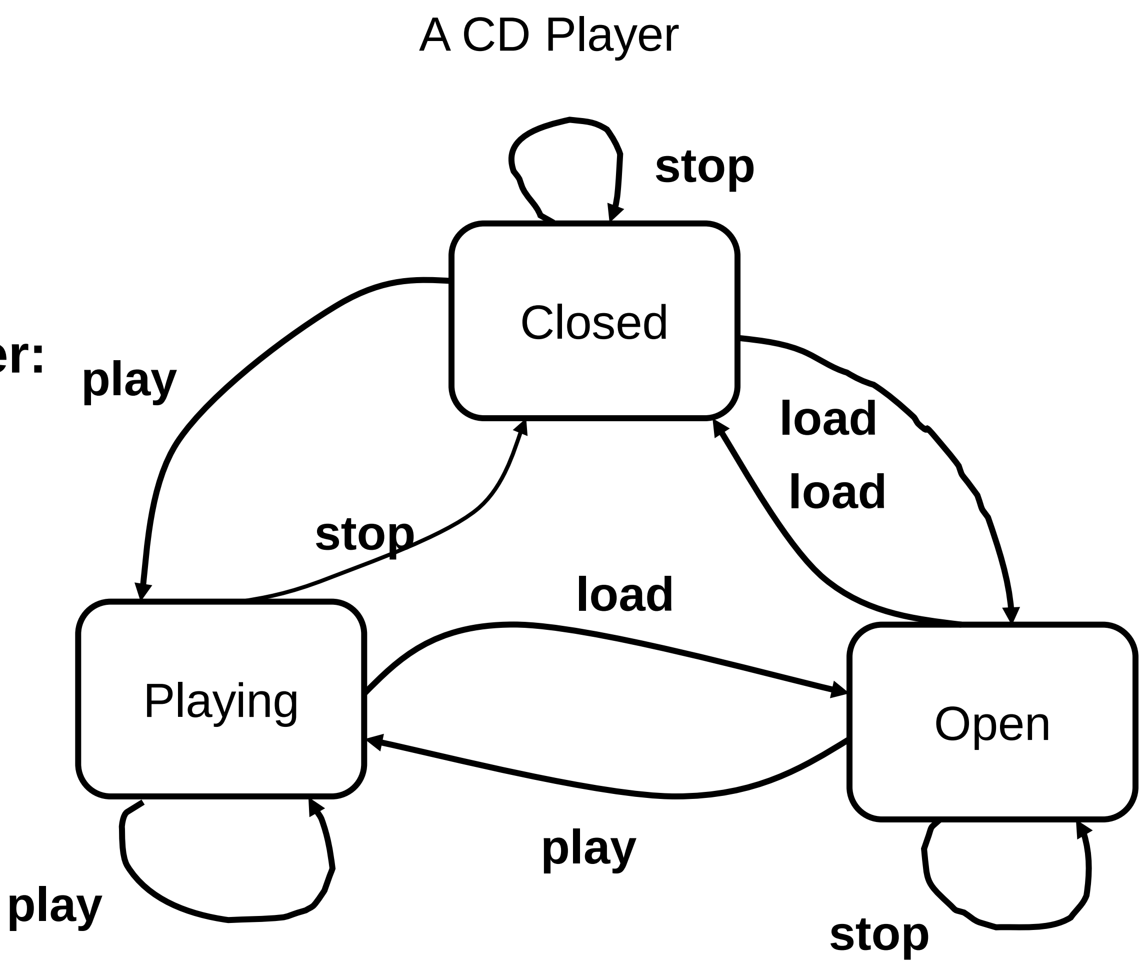
How to Model a State Machine Diagram

1. Identify the entities (objects or use cases) that need to be further detailed.
2. Identify the start and end states for each entity
3. Determine the events relating to each entity
4. Create state machine diagram beginning with the start event
5. Create composite states where necessary

Example - CD Player

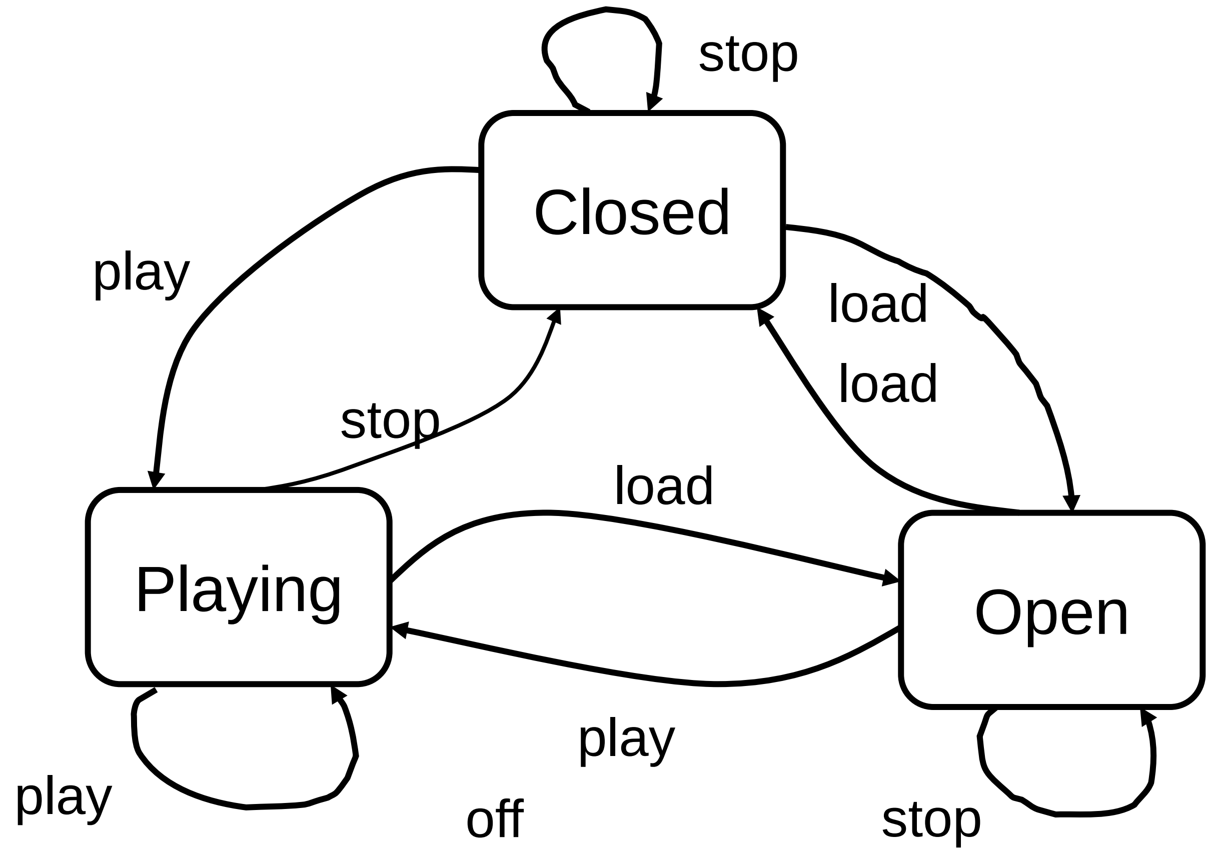
- **CD Player has 3 states:**
  - Playing
  - Opened
  - Closed

- **Events for the CD player:**
  - Load
  - Stop
  - Play

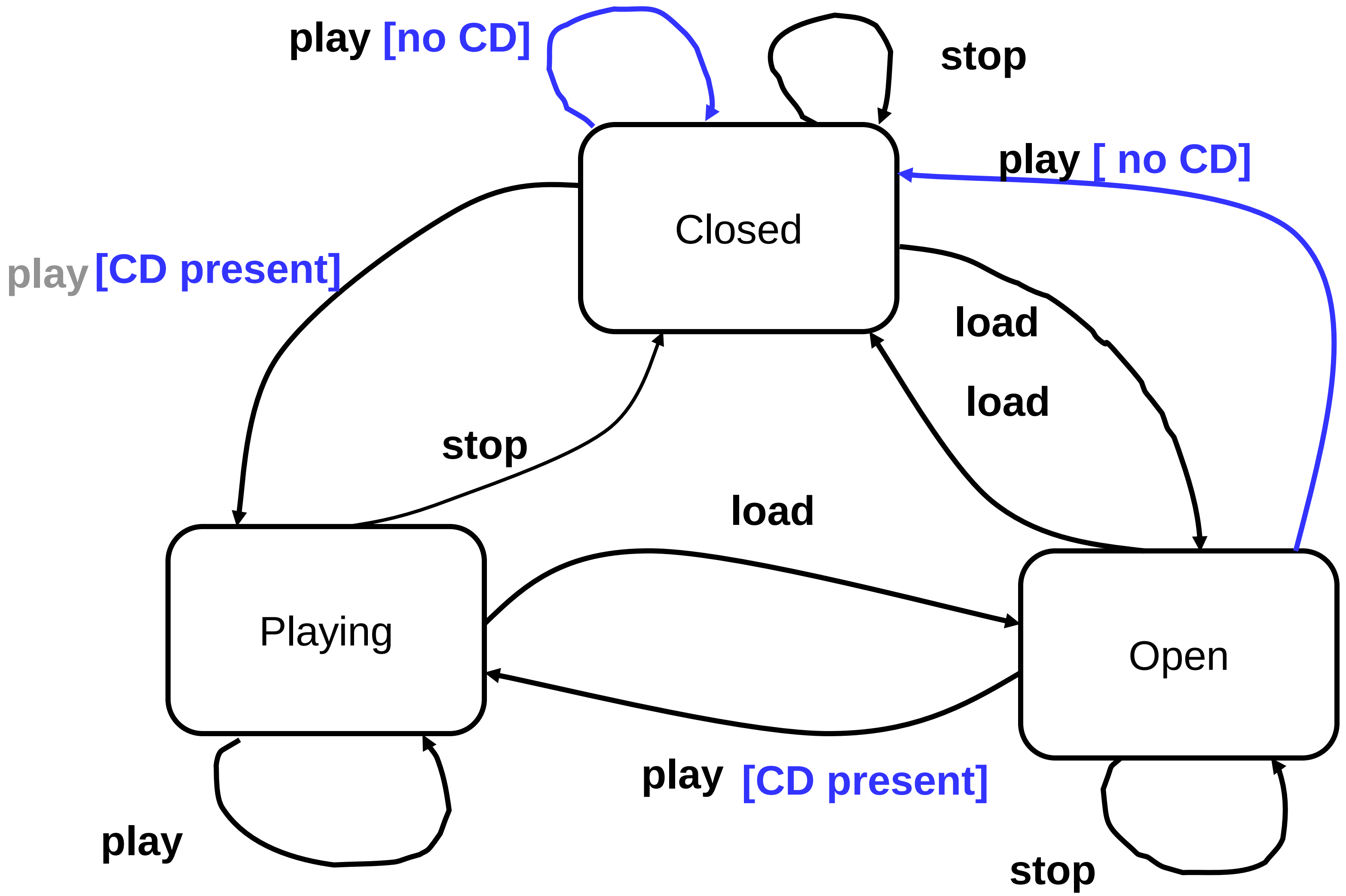


Who decides these?

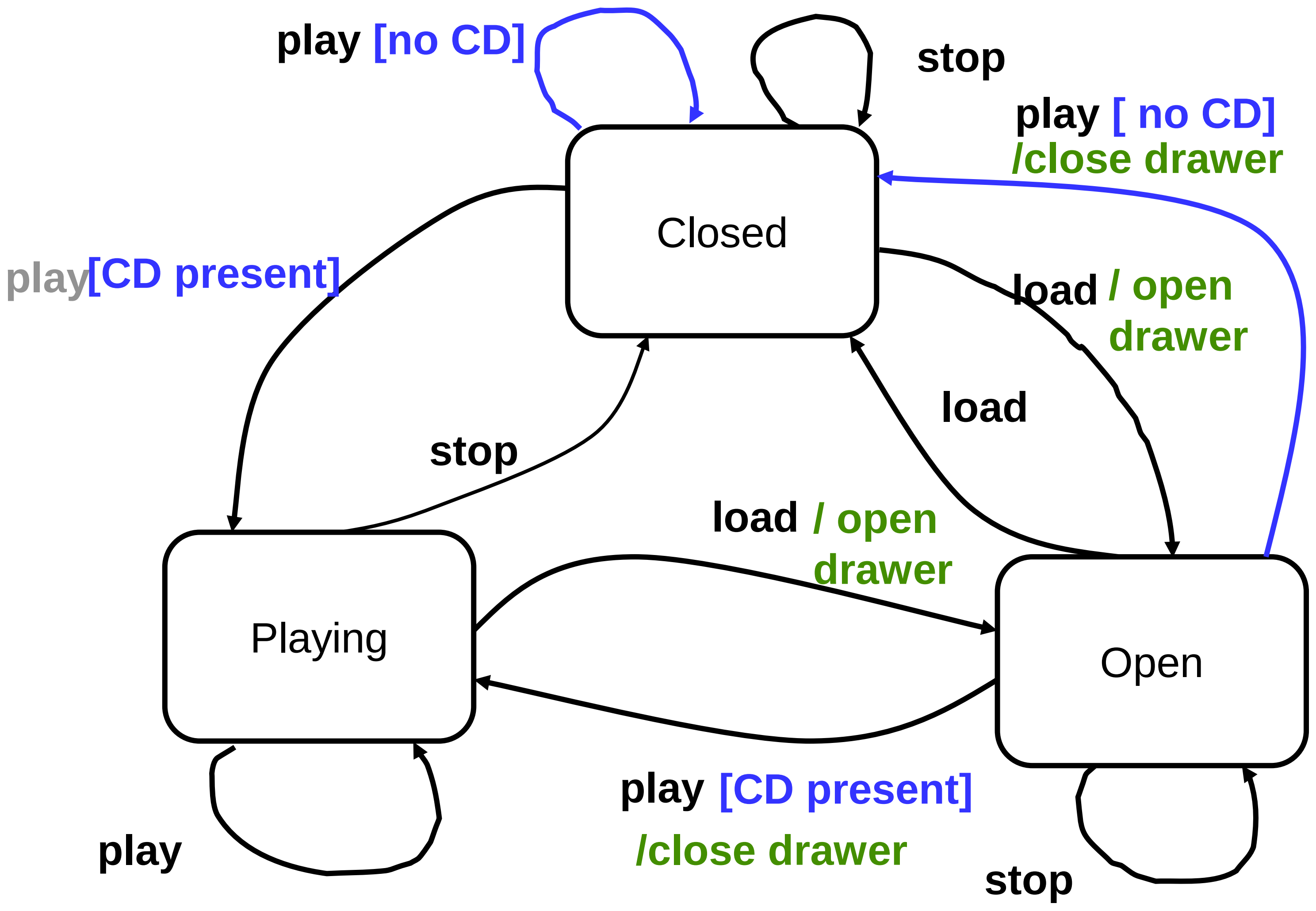
Example - initial and final states



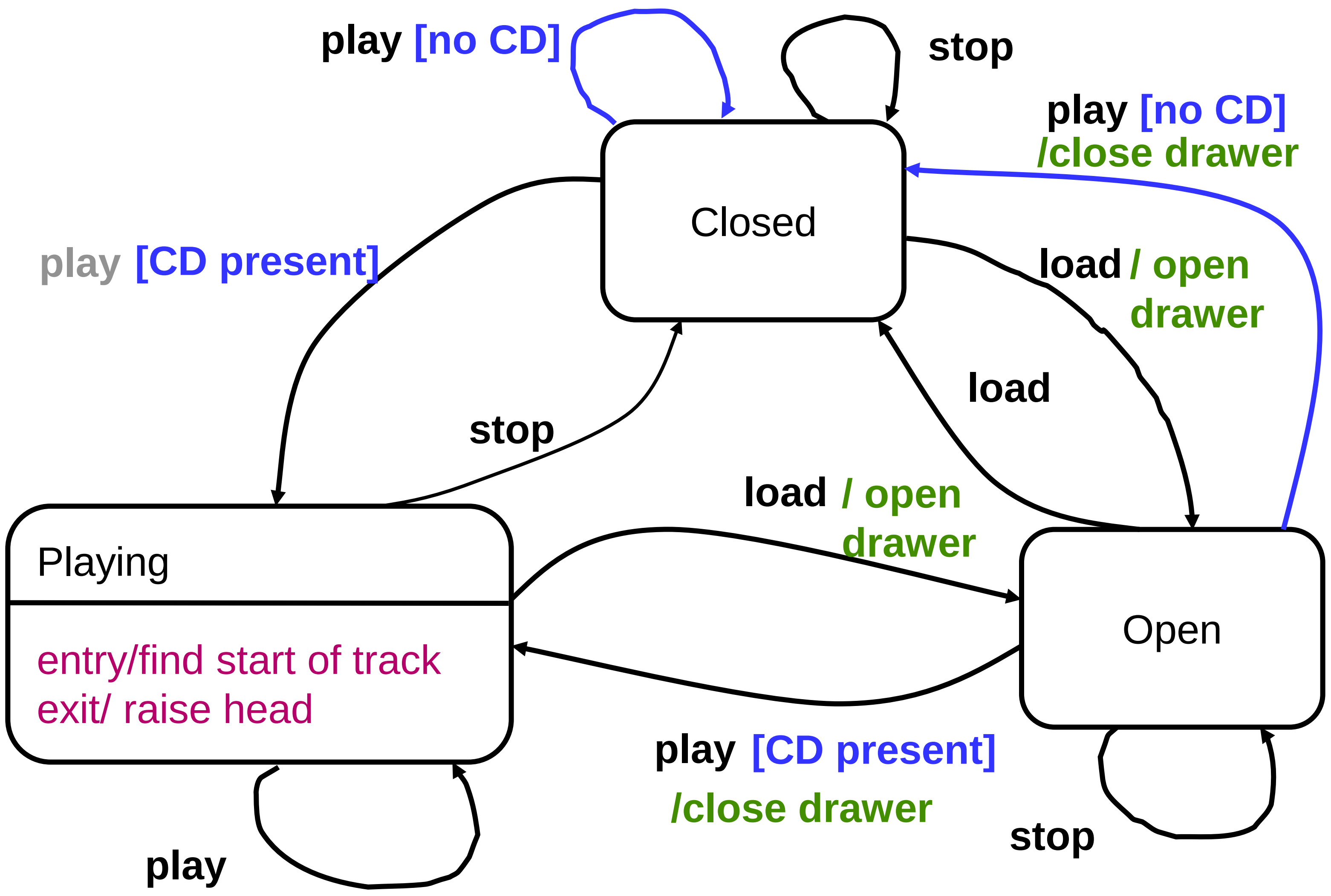
A CD Player – Guard Conditions



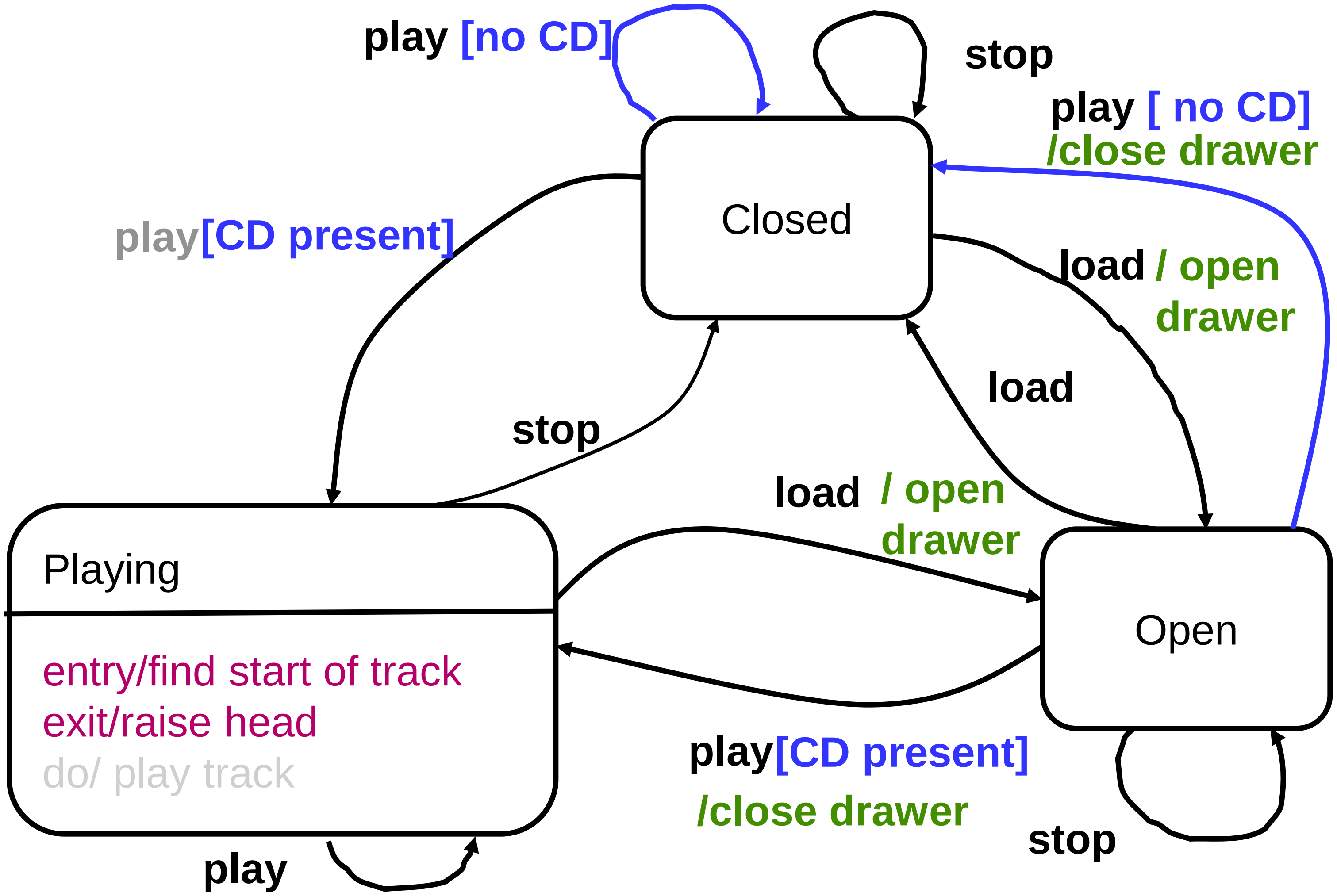
A CD Player – Actions



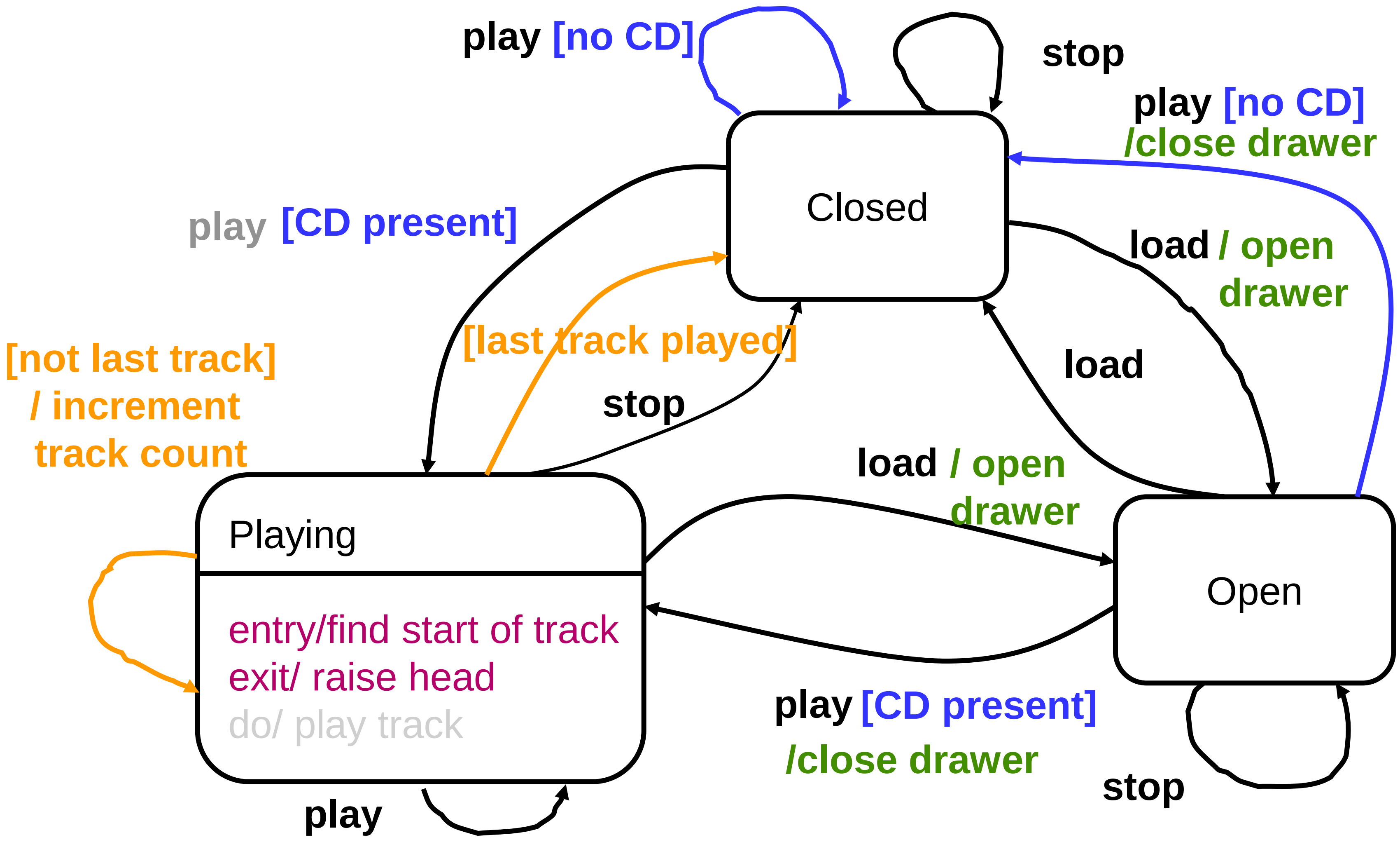
A CD Player – Entry /Exit actions



A CD Player – Activities



A CD Player – Internal Transitions

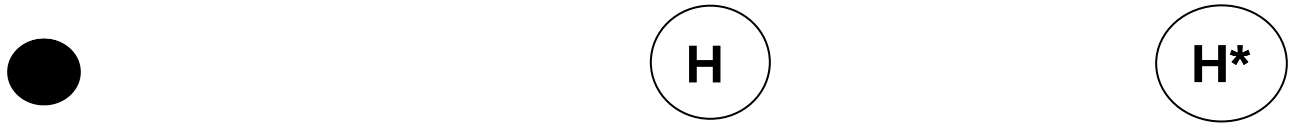


3. Advanced Topics

State Machine Diagram

- Composite state with entry/exit points: increase the scalability and independence of behavior specification.
- State machine generalization: enables inheritance and specialization of behavior.
- Protocol state machines: specifies allowed sequences of signals and operation calls

Pseudo States



Initial Pseudo state · Shallow History · Deep History

- *Pseudostate:* an abstraction that encompasses different types of transient vertices in the state machine graph.

- Transition from an initial pseudostate: may be labeled with the trigger event that creates the object. If unlabeled, it represents any transition from the enclosing state.

- *DeepHistory*: The containing state does not have to be exited in order for deepHistory to be defined.

- *ShallowHistory:* represents the most recent active substate of its containing state (but *not* the substates of that substate).

Entry Point and Exit Point

name ○ ⊗ name

- UML1.4 has no limitations on how to enter and exit composite states.
- It is legal to enter directly to internal states of composite states and it is therefore difficult to specify composite states that can be reused in other state machines.
- UML2.0 has introduced exit and entry points that control access to a composite state.

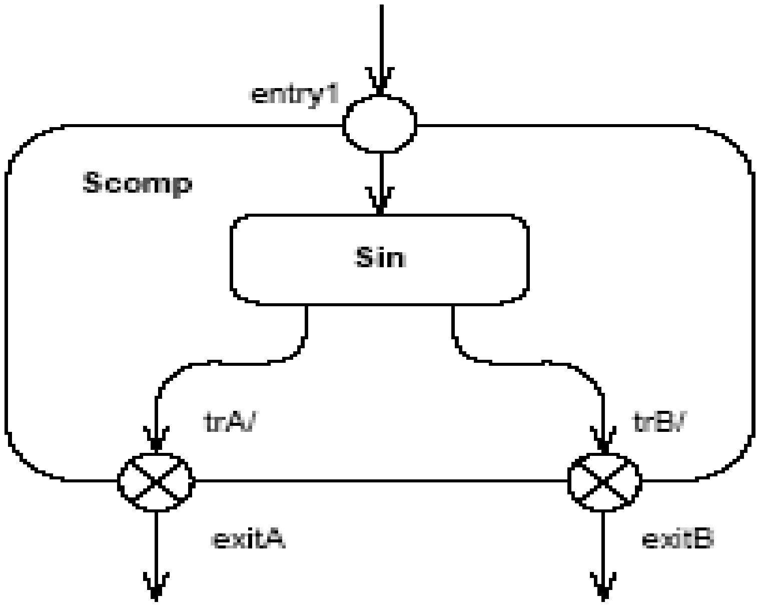


Figure 15.21 - Entry and exit points on composite states

Entry Point and Exit Point

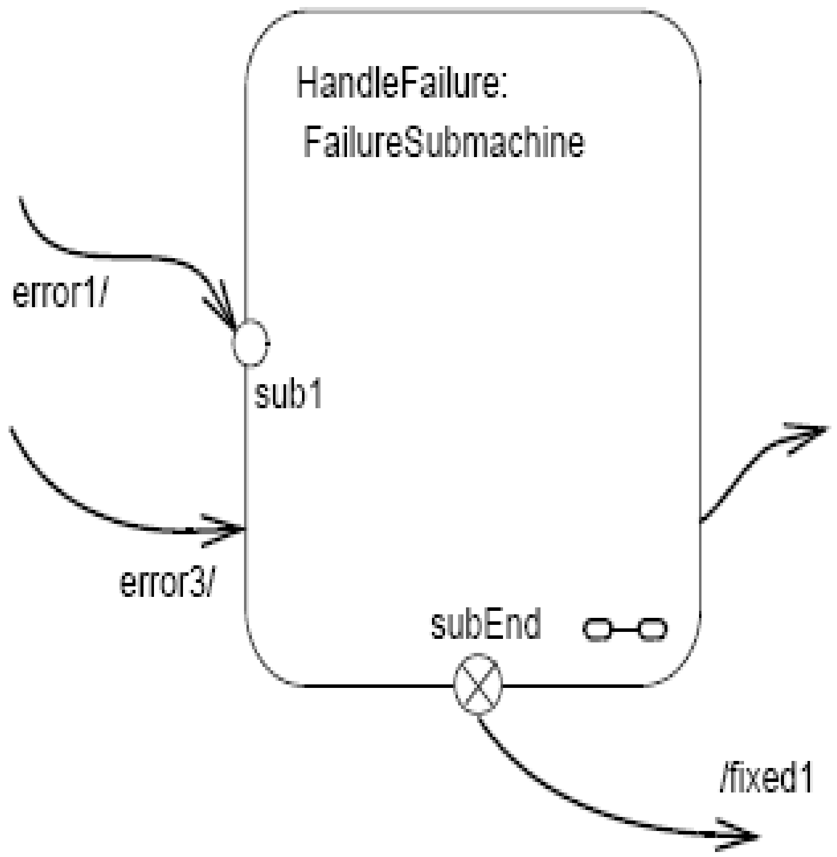


Figure 15.36 - Submachine State

- FailureSubmachine is referenced.
- Transition triggered by "error1" will terminate on entry point "sub1".
- Transition emanating from "subEnd" will execute "fixed1" in addition to what is executed within HandleFailure.
- "error3" transition implies taking of the default transition of FailureSubmachine.
- Transition emanating from the edge of the submachine state is the result of the completion event generated when the FailureSubmachine reaches its final state.
- The same notation applies to composite states with the exception that there would be no reference to a state machine in the state name.

Entry Point and Exit Point

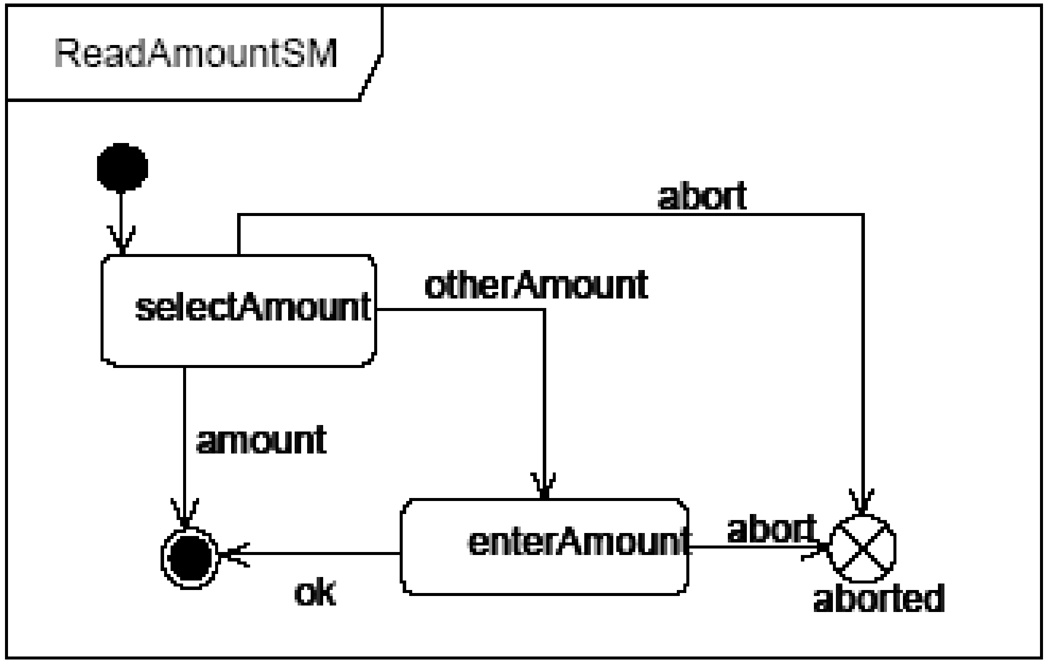


Figure 15.37 - State machine with exit point as part of the state graph

- Defined with two exit points.
- The entry and exit points may also be shown on the frame or outside the frame (but still associated with it), and not only within the state graph.

Junction and Choice Pseudo State

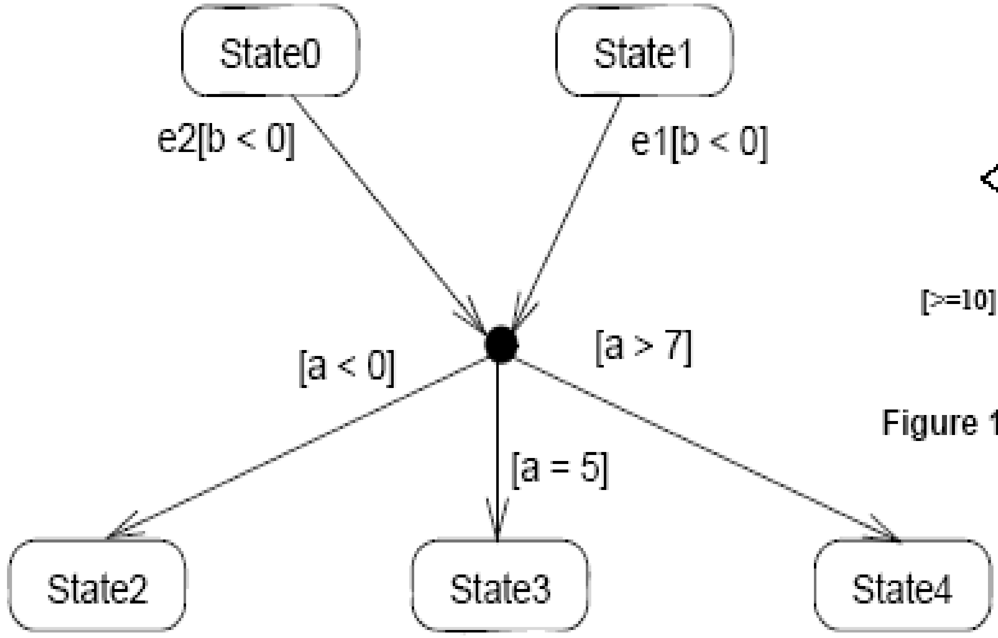


Figure 15.22 - Junction

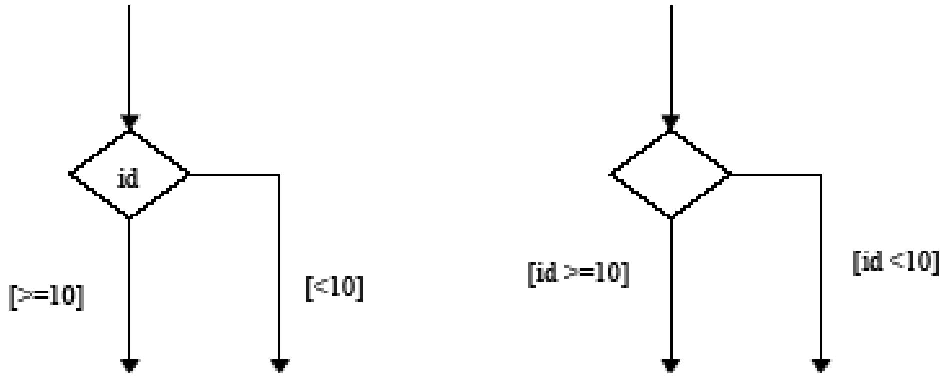


Figure 15.23 - Choice Pseudo State

Generalization of Behavior

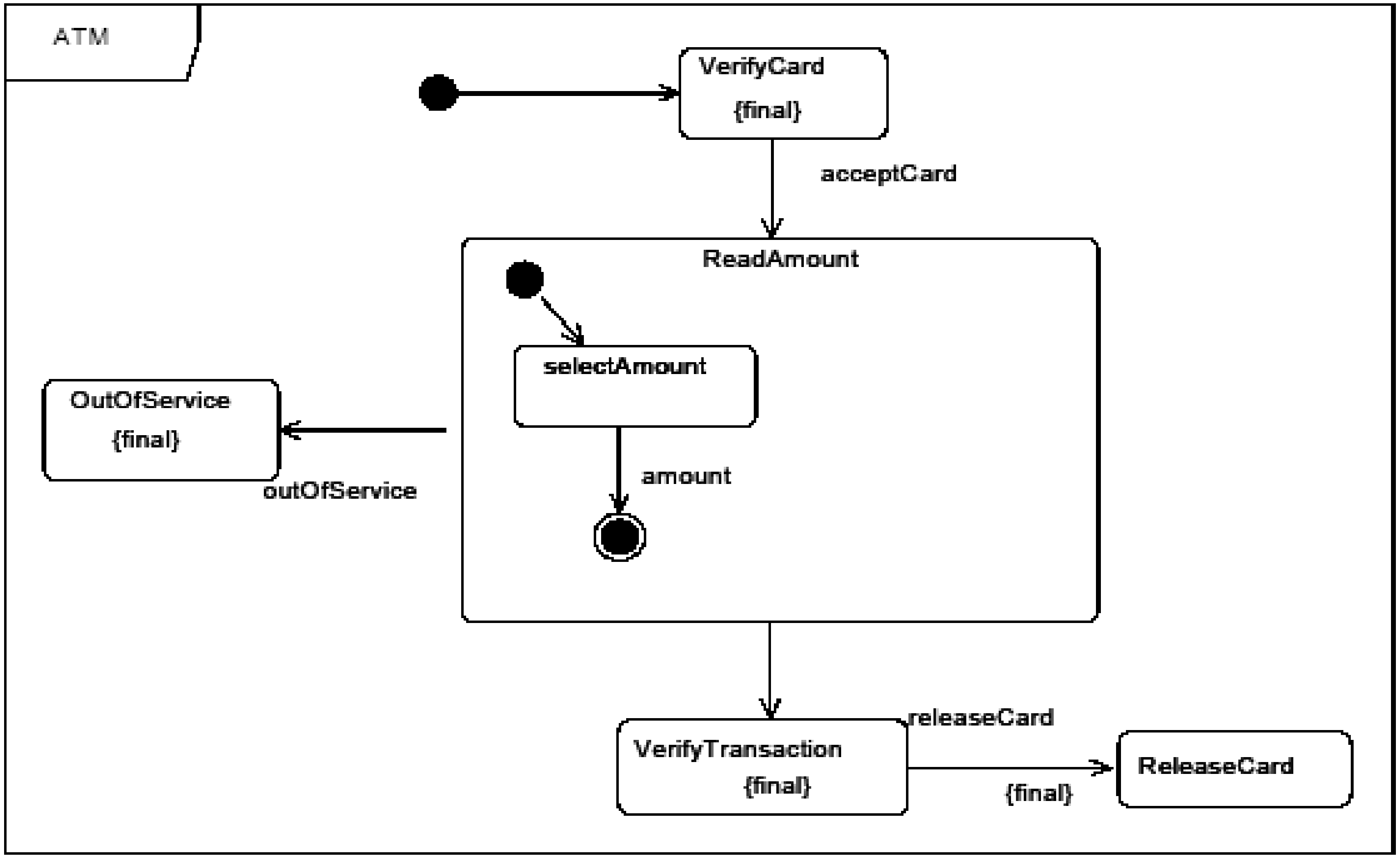


Figure 15.42 - A general state machine

Generalization of Behavior

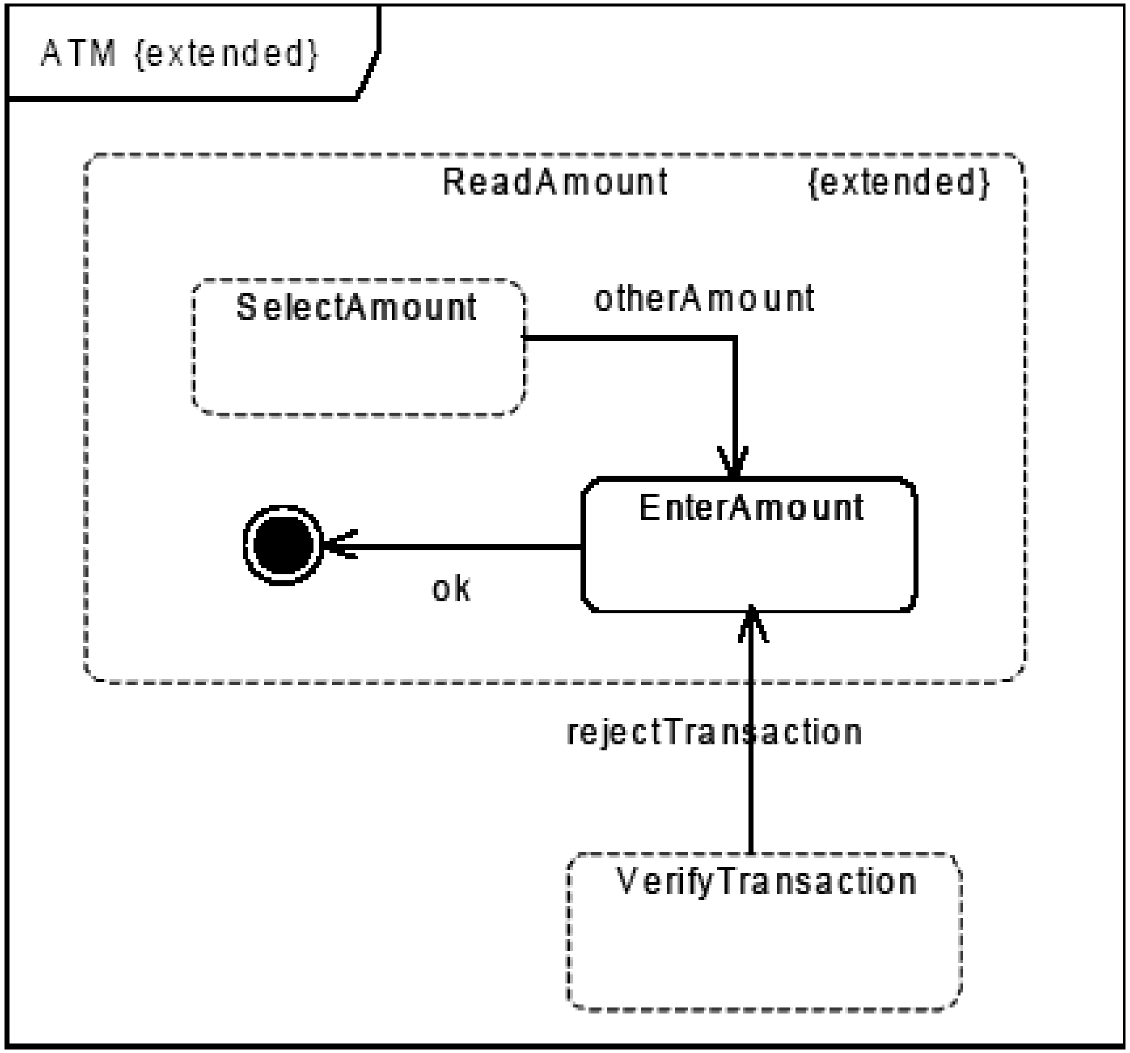


Figure 15.43 - An extended state machine

- New state machine types can also be specified using inheritance independent of classes.
- New behavior specification can be added and parts of existing behavior can be replaced.

Generalization of Behavior

- Figure 15.43 is a specialization of Figure 15.42.
- It is defined by extending the composite state by adding a state and a transition, so that users can enter the desired amount.
- In addition a transition is added from an inherited state to the newly introduced state.
- The composite state *ReadAmount* may be extended adding one new state (*EnterAmount*), three transitions (*OtherAmount*, *ok, rejectTransaction*)

Protocol State Machine

- Specifies which operations of the classifier can be called in which state and under which condition.
- May be attached to an interface, port, and to the component itself
- Defines the external view more precisely by making dynamic constraints in the sequence of operation calls explicit.
- Other behaviors may also be associated with interfaces or connectors to define the ‘contract’ between participants in a collaboration (e.g., in terms of use case, activity, or interaction specifications).
- Just like a normal state machine but cannot have entry or exit actions, activities, internal transitions history states and so on.

Protocol State Machine

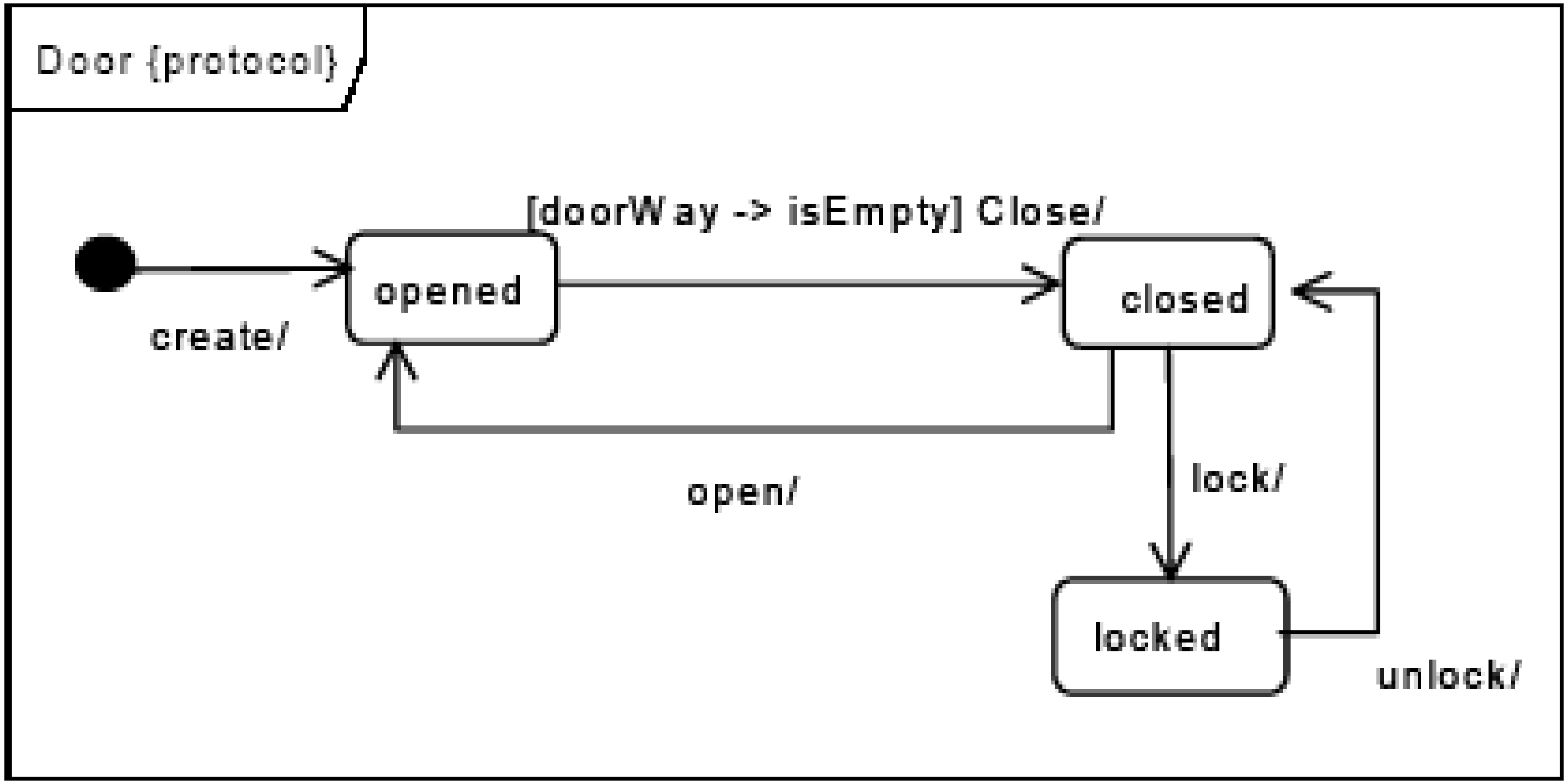


Figure 15.12 - Protocol state machine

Questions?