Using Enums to Implement State Machines

Jauhar Ali

https://scialert.net/fulltext/?doi=jse.2010.215.230

Air-Condition (AC) Controller

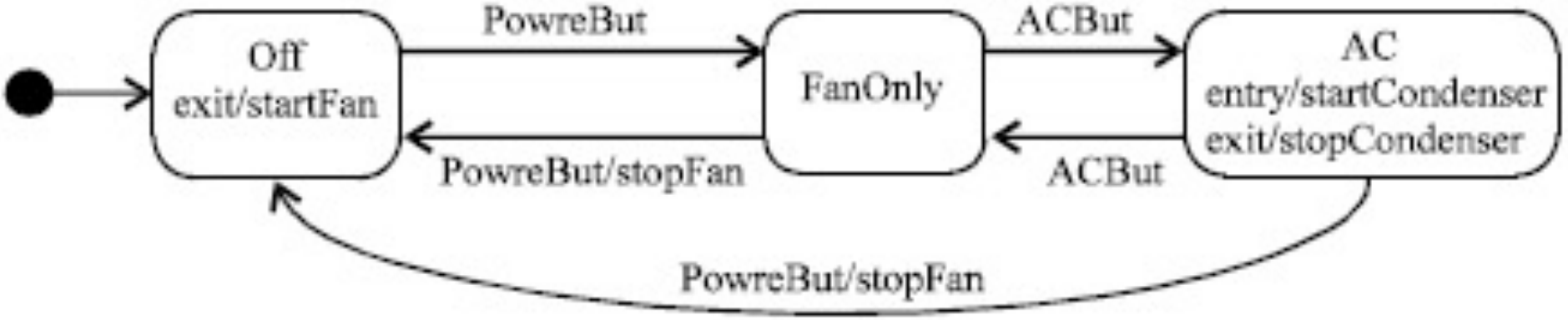
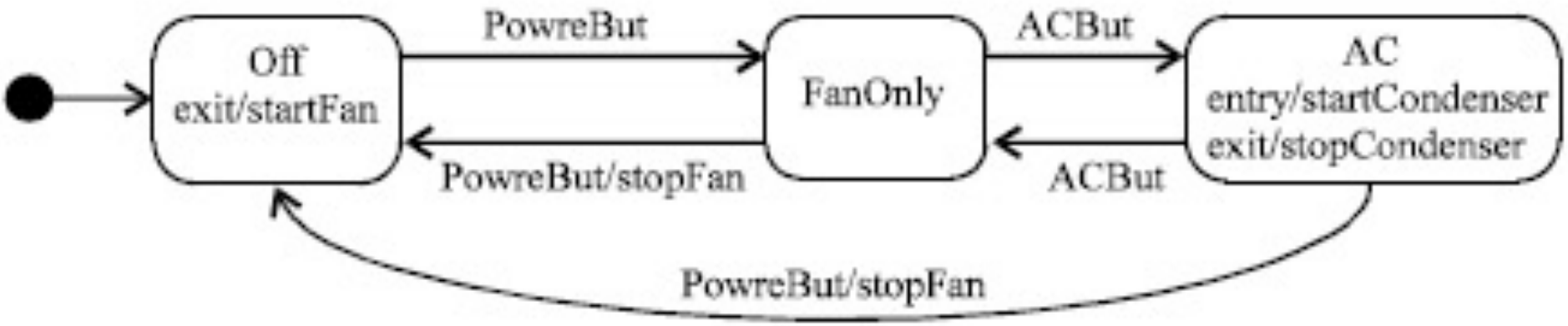
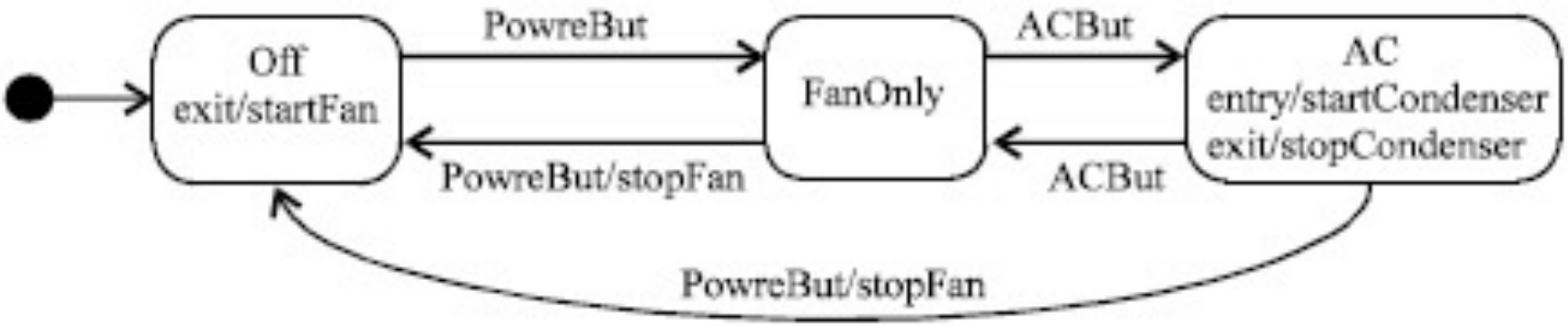


Fig. 1: Simple state machine diagram for air-condition controller



- The controller will be in one of three states: Off (default), FanOnly and AC.
- If Off is the current state and the PowerBut event occurs, the state will change to FanOnly.
- Similarly, if FanOnly or AC is the current state and the PowerBut event occurs, the stopFan action will execute and the state will change to Off.



- The ACBut event will change the state from FanOnly to AC and vice versa.
- Whenever the AC state is entered, the startCondenser action will execute.
- Similarly, whenever the AC state is exited, the stopCondenser action will execute.

To implement the AC Controller's state machine

use a nested class, called StateMachine, inside the ACController class.

- The StateMachine class encapsulates almost all aspects of the state machine.
- All actions in the state machine become methods in the ACController class (lines 9 - 24).
- The ACController and the StateMachine classes have references to each other (lines 2 and 35), through which they can call each other's methods.
- All events received by the ACController are delegated to the StateMachine (lines 27 - 32).

```
public class ACController
    StateMachine stateMachine;

    ACController() {
        stateMachine = new StateMachine(this);
    }

    // Action methods
    private void startCondenser() {
        // To be replaced with appropriate code
        Console.WriteLine("startCondenser executed");
    }
    private void stopCondenser() {
        // To be replaced with appropriate code
        Console.WriteLine("stopCondenser executed");
    }
    private void startFan() {
        // To be replaced with appropriate code
        Console.WriteLine("startFan executed");
    }
    private void stopFan() {
        // To be replaced with appropriate code
        Console.WriteLine("stopFan executed");
    }
```

```
26      // Events delegated to StateMachine
27      public void powerBut() {
28          stateMachine.powerBut();
29      }
30      public void acBut() {
31          stateMachine.acBut();
32      }
33
34      class StateMachine {
35          ACController context;
36          State state;
37
38          StateMachine(ACController context) {
39              this.context = context;
40              state = State.Off; // default
41          }
42
43          private void powerBut() {
44              state.process(this, Event.PowerBut);
45          }
46          private void acBut() {
47              state.process(this, Event.ACBut);
48          }
```

To implement the AC Controller's state machine

- Inside the StateMachine class, we use two enums:
- Event (line 51) and State (line 54).
- The Event enum represents all events and the State enum represents all states in the state machine.
- Each event and state becomes an enum value. For example, Off (line 55) and FanOnly (line 68) become enum values inside State.
- The state (line 36) reference inside StateMachine represents the current state of the state machine.

```
50          // All events
51          enum Event { PowerBut, ACBut }
52
53          // All states
54          enum State {
55              Off {
56                  void exit(StateMachine sm) {
57                      sm.context.startFan();
58                  }
59                  void process(StateMachine sm, Event event) {
60                      switch(event) {
61                          case PowerBut:
62                              this.exit(sm);
63                              sm.state = FanOnly;
64                              sm.state.entry(sm);
65                      }
66                  }
67              },
```

To implement the AC Controller's state machine

- Enums can have methods and data members. Each enum value can override the methods.
- The State enum has empty entry (line 106) and exit methods (line 107) .
- The AC state (line 83) overrides these methods because it has entry and exit actions in the state machine.
- The State enum has also an abstract method, named process (line 105), which is overridden by all states.
- It is called by the StateMachine on the current state whenever an event is delegated to the StateMachine (lines 43 and 46).

```
68          FanOnly {
69              void process(StateMachine sm, Event event) {
70                  switch(event) {
71                      case ACBut:
72                          this.exit(sm);
73                          sm.state = AC;
74                          sm.state.entry(sm);
75                          break;
76                      case PowerBut:
77                          this.exit(sm);
78                          sm.context.stopFan();
79                          sm.state = Off;
80                          sm.state.entry(sm);
81                  }
82          },
```

```
83          AC {
84              void entry(StateMachine sm) {
85                  sm.context.startCondenser();
86              }
87              void exit(StateMachine sm) {
88                  sm.context.stopCondenser();
89              }
90              void process(StateMachine sm, Event event) {
91                  switch(event) {
92                      case ACBut:
93                          this.exit(sm);
94                          sm.state = FanOnly;
95                          sm.state.entry(sm);
96                          break;
97                      case PowerBut:
98                          this.exit(sm);
99                          sm.context.stopFan();
100                         sm.state = Off;
101                         sm.state.entry(sm);
102                 }
103         };
```

To implement the AC Controller's state machine

- All transitions from a state are implemented in the process method for that state.
- The process method takes an event as parameter and chooses one case from the switch statement depending on the event.
- Each case corresponds to one transition. For example, the first case in the process method of the FanOnly state implements the transition on the ACBut event (line 71).
- Inside each case (which corresponds to a transition), three methods are called in the given order: (1) the exit method of the current state, (2) the action method (if any) for the transaction and (3) the entry method of the new state.

```
105                 abstract void process(StateMachine sm, Event event);
106                 void entry(StateMachine sm);
107                 void exit(StateMachine sm);
108             }
109         }
110     }
```