Implementation of Finite State Machine for Temperature Control System

Nyein Zarni Wai
University of Computer Studies, Yangon
alwaysforu09@gmail.com

Abstract

Finite State Machines are found throughout computer science. Compilers, grammars, or any kind of program where users can be in different positions, all make use of high-level states and state transitions. Any circuit with memory is a Finite State Machine (FSM). Even computers can be viewed as huge FSMs. The Finite State Machine is a useful design tool that allows us to approach the design in a systematic manner. Finite state machines are widely used in modeling of application behavior, design of hardware digital system, software engineering, compilers, network protocols, and the study of computation and languages. This system develops software designs using event-driven finite state machine and implements on temperature control system as a case study.

Keywords: Finite State Machine (FSM), Temperature Control System, Event-driven FSM.

1. Introduction

The most difficult area of software for embedded systems, for telecommunications, and for "reactive systems" in general is to ensure that the behaviour of the software in all circumstances is well controlled, including the presence of unusual combinations of external stimuli and of errors. This area is best expressed in terms of finite state machines (FSM), which are much easier to design and understand.

The State Machine, or more accurately, Finite State Machine (FSM), is a device and a technique that allows simple and accurate design of sequential logic and control functions. A finite state machine is simply a machine that has a finite number of states.

Many mechatronics systems have elements of both: some parts have infinite numbers of states while other parts operate as finite state machines with a limited number of possible states. The first step in designing or analyzing a finite state machine is to find all the possible states that it can be in. Sometimes it is possible to identify states that do not actually exist or are not required. These are called "redundant" states.

Software interfaces with the controlled application exchanging data: supplying input data and receiving output data. There are two basic interface mechanisms used: polling and event driven. Polling means that inputs are cyclically checked and outputs are cyclically set. Event driven means that the interface generates events if inputs change; outputs are also set if their values change. Event-driven systems are favored in software as a more effective.

Designers used truth tables to represent relationships among the inputs, outputs, and states of a finite state machine. The resulting table describes the logic necessary to control the behavior of the system. The first step in any FSM design is to identify the significant states of the system; all the important states are needed to include. This document describes a way in which the FSM concept can be applied in software, and by which FSM designs can be expressed in full detail in XML format.

2. Related Work

The authors from [1] showed that a state machine should perform the complete control function, as regards behavior. This is the most important rule for state machine design. If this rule is not kept, the final control system that is partly realized by a state machine and partly elsewhere in the code does not make sense. Unfortunately, coded implementations of control systems tend to ignore this issue, which explains the limited usage of state machines in software.

Thomas Wagner [2] introduced that as the FSM functions, changing state from time to time, it will provoke actions in other parts of the system, as required for the specific project. The actions can be performed by entering a state (entry actions), leaving the state (exit actions) or they can be triggered by an input (input actions) irrespectively of the state transition. Input actions and entry actions are the basic actions used by state machine specification. A state machine which uses only entry actions is called a Moore model. A state machine which uses only input actions is called a Mealy model. In practice, models which combine all actions: entry, exit and input are preferable solutions of state machines. This

document describes a way in which the FSM concept can be applied in software, and by which FSM designs can be expressed in full detail in XML format. A detailed description about FSM and VFSM can be found.

David Gibson [3] said that one of the most fascinating things about FSMs is that the very same design techniques can be used for designing Visual Basic programs, logic circuits or firmware for a microcontroller. Many computers have at their hearts, an FSM. In disciplines other than engineering and programming FSM concepts are used for pattern recognition, artificial intelligence studies, language and behavioral psychology. And yet, the basic concepts are easy to understand and immensely powerful. The idea behind the FSM is that a system such as a machine with electronic controls can only be in a limited (finite) number of states. Consider some simple systems that you encounter every day: a door may be open or closed; a light may be on or off; a light bulb may be on, off or broken.

3. Background Theory

A finite state machine (FSM) or finite state automaton (plural: automata) or simply a state machine, is a model of behavior composed of a finite number of states, transitions between those states, and actions. All states represent all possible situations in which the state machine may ever be. Hence, it contains a kind of memory: how the state machine can have reached the present situation. As the application runs the state changes from time to time, and outputs may depend on the current state as well as on the inputs. A control system determines its outputs depending on its inputs. If the present input values are sufficient to determine the outputs, the system is a combinational system, and does not need the concept of state. If the control system needs some additional information about the sequence of input changes to determine the outputs, the system is a sequential system. The true task of state machine is to trigger actions according to situation defined by the present state and stimuli. A transition indicates a state change and is described by a condition that would need to be fulfilled to enable the transition. An action is a description of an activity that is to be performed at a given moment. There are several action types: **Entry action** which is performed when entering the state. **Exit action** which is performed when exiting the state. **Input action** which is performed depending on present state and input conditions. **Transition action** which is performed when performed a certain transition.

Each State has its state transition table. The table consists of several fields used to specify actions and transitions.

State	Entry action	Entry_Action1 Entry_Action2
	eXit action	Exit_Action1 Exit_Action2
	Input_Action_Condition1	Input_Action1 Input_Action2
	Input_Action_Condition2	Input_Action3
Next_State1	Transition_Condition1	
Next_State2	Transition_Condition2	

Figure 1. Example of State Transition Table

In the figure.1 shows the state transition table for the finite state machine.

3.1 Event-Driven FSM

If the software control system uses a state machine model to describe the system behavior, its model should not be determined by the input acquisition system but rather by the application requirements. In some software implementations of state machines a purely event-driven state machine model, which implies that the state machine has to store not only the history of state changes but also the actual present value of inputs.

It is obvious that an FSM operates as a result of various changes to its input, and this leads to the assumption that an FSM is "event driven" – which is in a sense true. Each incoming event is "consumed" by the FSM as it operates. An event which does not cause a transition is discarded in these schemes, and this leads to a need to store events which might be needed in future, by ensuring that they always cause a transition to a new state. An FSM must have access to all the inputs which will determine its transitions, at the instant when any transition expression is being evaluated.

3.2 Classification

There are two different groups: Parsers and Transducers.

3.2.1 Parsers

A parser state machine is also called a recognizer or acceptor. Such a state machine has an initial and a final state and its path from the start to the final state is deterministic; i.e., there is only one transition from each state.

3.2.2 Transducers

Transducers generate output based on a given input and or a state using action. They are used for control applications and in the field of computational linguistics. Two types are distinguished as follow:

3.2.2.1 Moore machine

The FSM uses only entry actions, i.e., output depends only on the state. The advantage of the Moore model is a simplification of the behaviour. At an elevator door, the state machine recognizes two commands: "command_open" and "command_close" which trigger state changes. The entry action (E:) in state "Opening" starts a motor opening the door, the entry action in state "Closing" starts a motor in the other direction closing the door. States "Opened" and "Closed" don't perform any actions. They signal to the outside world (e.g., to other state machines) the situation: "door is open" or "door is closed".

3.2.2.2 Mealy machine

The FSM uses only input actions, i.e., output depends on input and state. The use of a Mealy FSM leads often to a reduction of the number of states. An elevator door has two input actions (I:): "start motor to close the door if command_close arrives" and "start motor in the other direction to open the door if command_open arrives". The "opening" and "closing" intermediate states are not shown.

If the output function is a function of a state and input alphabet () that definition corresponds to the Mealy model, and can be modelled as a Mealy machine. If the output function depends only on a state () that definition corresponds to the Moore model, and can be modelled as a Moore machine.

3.3 State Machine Models Behavior

A state machine is a model of behavior; in general it models a control system that is to supervise an application. This definition means that a state machine is a decision machine that generates signals representing actions: do this or do that. A given state machine reflects the ability of the designer to translate an informal control specification into a formal logical model in a form of a state machine.

The best results are achieved by combining both models: Mealy and Moore. For a hardware implementation this kind of decision is not so simple because it may mean additional expense. For a software implementation the choice of a state machine model has no financial consequences — it influences only the design process.

A combined Mealy/Moore model means that the important actions strongly linked with a state should be specified as Entry Actions. This rule underlines the character of the actions: Entry Actions are state dependent, Input Actions are input (and present state) dependent.

3.4 FSM Logic

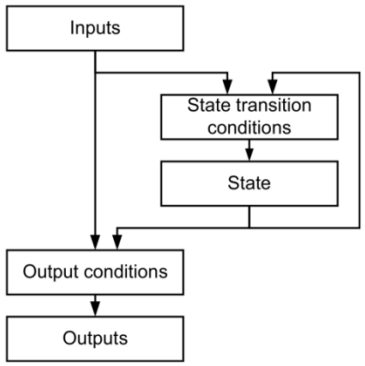

Figure 2. State machine definition

4. System Overview

The FSM Controller is the utility and easily port to every Java application in Finite state machine enable system. And this system generates the finite states that contain its entry actions, exit actions, and input actions which depend upon current state conditions and transitions for current state to other state. Figure 3 and 4 show the flow control of the action and condition for the temperature control system.

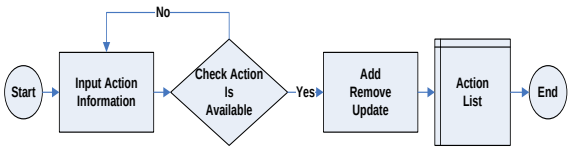

Figure 3. The flow control of the Action available for the System

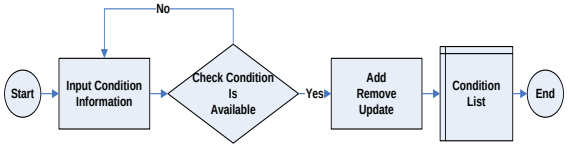

Figure 4. The flow control of the Condition available for the System

This system has three main parts for utilizing for Finite state enable application. There are:

1. FSM Generator
2. FSM Extractor
3. FSM Runner.

4.1 FSM Generator

This module is a one part of the FSM Controller system that is utilized for generating FSM database in XML format in figure 5. FSM database can be defined the DEVICEs and, its action and primitive conditions for possible circumstance for intended fields or industrials. Thereafter it also be defined the STATEs that has entry and exit actions, and its transitions.

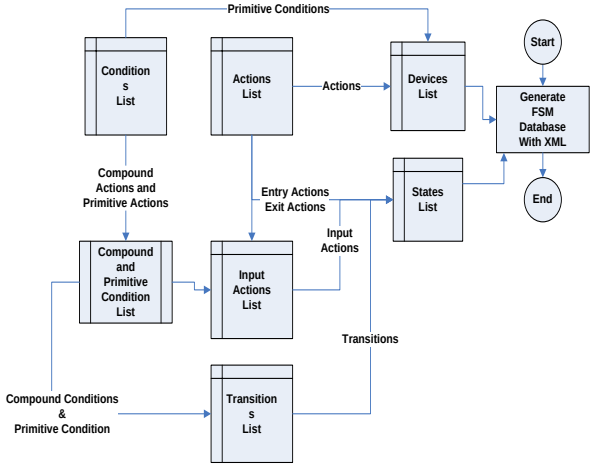

Figure5. Flow control of the generated FSM Database with XML

4.1.1 Generating FSM Database

```
<?xml version="1.0" encoding="UTF-8"
standalone="no"?>
<fsm name="temp">
 <device>
<device-name>Aircon</device-name>
      <device-type>
controller.device.AirConditioner</device-type>
      <description/>
      <action>
            <action-name>DoAirconOn
          </action-  name>
            <action-type>
controller.action.AirconOnAction</action-type>
     <description/>
     </action>
     <action>
            <action-name>DoAirconOff
          </action-name>
            <action-type>
controller.action.AirconOffAction</action-type>
     <description/>
     </action>
     <condition>
            <condition-name>IsAirconOn
          </condition-name>
            <condition-type>
controller.condition.AirconOnCondition
            </condition-type>
    <description/>
   </condition>
```

```
        </condition-name>
```

```
<condition>
            <condition-name>IsAirconOff
            </condition-name>
            <condition-type>
controller.condition.AirconOffCondition
            </condition-type>
    <description/>
    </condition>
</device>
```

Figure 6. Part of FSM database

Part of the generated database XML from the FSM Controller utility software is shown in figure 6.

4.2 FSM Extractor

This module is also a part of the FSM Controller system. That should be needed for extract instances of respective devices, their conditions and their actions. This module needs to extract the devices from the database, because of in the real world the device is just one instance for the whole application like that the printer instance is only one instance for all application in operating system. The FSM extractor extracts one instance of every device in database. And then that instance encapsulated in respective actions and condition which use in the FSM application.

4.3 FSM Runner

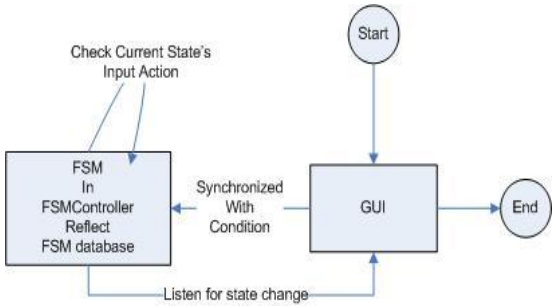

Figure 7. The flow control of the Application synchronized with the device and graphical user interface with FSM Controller.

Figure 7 shows that this module is utilizing for GUI and managing state to state upon its conditions and actions which were defined in the FSM database.

5. System Implementation

This paper presents the concept of air conditioner and heater control system using a finite state machine system.

5.1 Process flow of temperature control system

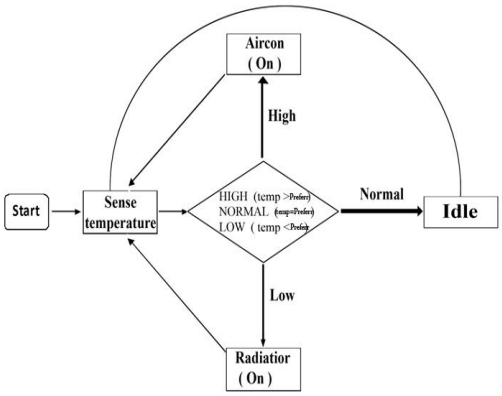

Figure 8. Process flow of temperature control system

In figure 8, the air conditioner and heater start and stop the heating and air conditioning system and are regulated by the temperature sensor which checks the temperature periodically ensuring temperature is within preferable level.

5.2 Checking the Temperature

The temperature sensor 'read' when the current temperature is greater than preferred temperature, would be reinterpreted as a 'temp_high'. And action 'air_on' would be reinterpreted as the 'start' event in the output state chart. Then the finite state machine changes to on. Entering state on, finite state machine switches air conditioner on. The temperature sensor 'read' when the current temperature is less than preferred temperature would be reinterpreted as 'temp_low' and action 'air_off' would be reinterpreted as the 'stop' event to the output state chart. And then, finite state machine changes its state to off. This is implementation for air conditioner. For heater, the implementation is vice-visa.

5.3 Temperature Control System

This control system has the following inputs:
Power push button – to start regulating when activated. **Temperature sensor** is to detect the temperature at preferable level.
The following outputs:
Air Conditioner - can be on or off,
Heater - can be on or off.

5.3.1 Defining States

In a state machine design the choice of the states is probably the most important decision. Theoretically, we should use only those states that are required for a given control (sequential) problem. There are five defined states in temperature control system.

5.3.1.1 Init State

The state machine begins with an initial state, which called Init. In the state Init the system does not work and the state machine waits for Power-on command. Table 1 shows that init state is the initial state. Once left it will never be reached again. After Init state, the state machine always changes to state Idle.

Table 1. State transition table for Init state

Init	Entry Action	
	Exit Action	Power_on
Idle	Always	

To activate the temperature controller, power button is pressed. After activated, the temperature firstly entering the sensor is current temperature. At first, we must define the preferred temperature and then temperature sensor detects current temperature to determine 'on' or 'off' the devices.

5.3.1.2 Idle State

In the state Idle all activities are ceased. All devices does not also work but still power-on. Waiting for set command. When the system is power on, the state machine changes to state Set. This consideration leads to a state transition table shown in table 2.

Table 2. State transition table for Idle state

Idle	Entry Action	DoAirconOff, DoHeaterOff
	Exit Action	DoSensorOff, Do Save
Set	IsPowerOn	

5.3.1.3 Set State

In state Set, several activities are initiated. Waiting for temperature acknowledgements. After Setting the preferred temperature, the state machine always change to next state Regulating. Table 3 describes the state transition of Set state.

Table 3. State transition table for Set state

Set	Entry Action	DoSensorOn, DoLoad
	Exit Action	
Regulating	Always	

5.3.1.4 Regulating State

In the state Regulating (table 4) where the state machine goes on receiving the command Set and temperature value is ok. The state Set is a "busy" state where the state machine sets the preferred temperature and waits for the reaction — if the pressure reaches the required temperature it will go to the "done" state (which we call Regulating). At state Regulating, entering the PowerOff makes returning to state Idle and the state machine waits for set command again. In case of failure: if the temperature is out of limit, next state is Error state.

Table 4. State transition table Regulating state

Regulating	Entry Action	
	Exit Action	DoSensorOff, DoHeaterOff DoAirconOff, DoSave
	IsOverPreferrTemp IsUnderPreferrTemp IsInPreferrTemp	
Idle	IsPowerOff	
Error	ErrCon*	

5.3.1.5 Error State

Temperature value is outside limit. If the temperature is too high or too low, the state machine goes to the state Error. Returning to state Idle is possible if power off. Table 5 shows the state transition of the Error state.

Table 5. State transition table for Error state

Error	Entry Action	TempTooHigh, TempTooLow
	Exit Action	
Idle	IsPowerOff	

6. Conclusion

The Finite State Machine (FSM) is an abstract concept that helps us analyze a mechatronics control system in a structured way. Finite state machines represent a very powerful way of describing and implementing the control logic for application to implement communication protocols and to control the interactions in a GUI, and many other applications. Using the control values for Transition and Input Action conditions and actions for outputs, software designers are able to specify in an abstract but complete way the behavior of the state machine neglecting details of implementation. This kind of state machine design is not completely implementation independent but these dependencies leave open the transformation between the real input/output signals and control values used by the state machine design. Many of these aspects can be modeled differently as determined by user requirements, and environment variables such as seasonal variation.

7. References

[1] Ferdinand Wagner, Ruedi Schmuki, Thomas Wagner, Peter Wolstenholme" Modeling Software with Finite State Machine" , A Practical Approach.

[2] Thomas Wagner, "Virtual Finite State Machine Mark-up Language", 2004.

[3] David Gibson, **"**Making simple work of complex functions", SPLat Controls Pty Ltd Pty Ltd.

[4] Gomez, M., "Embedded systems programming feature," *Embedded Systems* 13, no. 13 (December 2000).

[5] http://en.wikipedia.org/wiki/state_machine.

[6]http://en.wikipedia.org/wiki/Event-Driven_Finite_State_Machine.

[7] James Trevelyan, "Finite State Machine", School of Mechanical Engineering, based on lectures by Peter Kovesi, School of Computer Science and Software Engineering.