The International Journal Of Engineering And Science (IJES)
|| Volume || 3 || Issue || 5 || Pages || 54-58 || 2014 ||
ISSN (e): 2319 – 1813 ISSN (p): 2319 – 1805

Modeling & Control of Event Based Behaviour Using State Machines

[1]obota M.E. , [2]ukwa C.N. , [3]ona D.I.

[1]*Dept. of Physics, Ebonyi State College of Education, P.M.B. 02 Ikwo, Abakaliki, Ebonyi State, Nigeria.*
[2]*Dept of Computer Science, Ebonyi State College of Education, Ikwo, Abakaliki, Ebonyi State, Nigeria.*
[3]*Dept. Of Electrical/Electronic Engineering, Akanu Ibiam Fed. Polytechnic, Unwana, Ebonyi State, Nigeria.*

--ABSTRACT--
One of the main challenges in becoming an effective state machine designer is to develop a sense for which behaviour should be captured. In this paper, semantic of hierarchical state decomposition is deployed to facilitate reusing of behaviour to avoid unnecessary increase in system due to explosive increase in number of states and transitions. Furthermore, state machine is used here to model event based behaviour through transitions that are driven by the value of either discrete or continuous properties. Obviously, this research work is interesting especially to the software developers because the approach is truly applicable to real-life problems.

KEY WORDS: *State machine, modeling, transition, event, behavior,*

Date of Submission: 30 August 2014 Date of Publication: 25 September 2014

I. INTRODUCTION

Modeling and control of event based behaviour using state machines refers to model and control behaviour in terms of response of blocks to internal and external events, using state machine. State machines typically are used in system modeling and control to describe the state independent behaviour of block through its life cycle in terms of its state and the transitions between their states. System modeling language can be used to describe wide range of state-related behavior, say from the behaviour of simple lamp to switch, to the complex modes of an advance aircraft. For instance, many software are event-driven, which means that they continuously wait for the occurrence of some external or internal event such as mouse click, a button press, a time tick, or arrival of a data packet. After recognizing the event, such system reacts by performing the appropriate computation that may include manipulating the hardware or generating 'soft' events that trigger other internal software components (that is why event-driven systems are alternatively called reactive system). The response to an event generally depends on both the type of event and on the internal state of the system and can include a change of state leading to a state transition. The pattern of events states and transitions among the states can be abstracted and represented as state machine. The actions of state machine are similar to a flow graph in which one can expect the way logic runs when certain conditions are met. It has finite internal memory, an input feature that reads symbols in sequence, one at a time without going backwards, and output feature which may be in form of user interface, once the model is completed. State machine can solve a large number of problems, among which is electronic design automation, communication protocol design, parsing and other engineering applications. In biology and artificial intelligence research, state machines are sometimes used to describe neurological systems and in linguistics, to describe the grammars of natural languages [1 2].

II. BASIC CONCEPTS

State machine: State machine is a potentially reusable definition of some sate-independent behavior. State machine typically executes in the context of a block and event experienced by the block to cause state transition.

State machine diagram: State machine diagrams are sometimes refers to as state charts or sate diagram, but the actual name is SysML, meaning System Machine Language. The machine name is the name of the represented machine and is intended to describe the purpose of the diagram.

State: A state represents some significant conditions in the life of a block. Typically, it represents some changes in how the block responds to events. A state in a state machine can define all valid switch positions (i.e. state of the system). Each state may have entry and exit behaviors which are performed whenever the state may perform a **do activity** that executes once the entry behaviour has been completed and continues to execute unit complete or the state excited. A state is represented by a round-cornered box containing its name.

Transitions: A transition specifies how states change within a state machine. A state machine always runs to completion once a transition is triggered, which means that they are not able to consume another trigger event until the state machine has completed the processing of the current event. Switching from one state to another is called

state transition, and the event that causes it is called the triggering event or simply the trigger. In the keyboard for example, if the keyboard is in the 'default' state when the caps lock key is pressed, the keyboard will enter the 'caps locked' state. However, if the keyboard is already in the 'caps locked' state, pressing caps lock will cause a different transition from the 'caps locked' to the 'default' state. In both cases, pressing caps lock is the triggering event. In extended state machine (state machine supplemented with variables), a transition can have a guard which means that the transition can 'fire' only if the guard evaluates to TRUE [3].

Transition fundamental: A transition specifies how states change within a state machine. A state machine always runs to completion once a transition is triggered which means that they are not able to consume another trigger event until the state machine has completed the processing of the current event. Transition may include one or more triggers, a guard, and an effect.

Triggers: Triggers identify the possible stimuli that cause a transition and are associated with events.

Event: In the most general terms, an event is something that happens that affects the system. An event can have associate parameters, allowing the event instance to convey not only the occurrence of some interesting incidents but also quantitative information regarding that occurrence. Once generated, the event instance goes through a processing life cycle that can consist of up to three stages. First, the event instance is **received** when it is accepted and waiting for processing (e.g. it is placed on the event queue). Later, the event instance is

dispatched to the state machine, at which point it becomes the current event. Finally, it is consumed when the state machine finishes processing the event instance. A consumed event instance is no longer available for processing the event instance. Four main types of events are:

- **Signal event:---** indicate that a new asynchronous message has been received. A signal event may be accompanied by a number of arguments that can be used in the transition effect.
- **Time events:**--- indicate either that a given interval has passed since the current state was entered (relative), or that a given instant of time has been reached.
- **Change events**:--- indicate that some conditions have been satisfied (normally that some specific set of attribute values hold).
- **Call** events:--- indicate that an operation on the state machine's owing block has been requested. A call event may also be accompanied by a number of arguments.

Guard: The transition guard contains an expression that must be true for the transition to occur. The guard is specified using a constraint when an event satisfies a trigger, the guard on transition, if present, is evaluated. If the guard evaluates true, the transition is triggered, and if guard evaluates to false then the event is consumed with no effect. Moreover, guards can test the state of state machine using the operator **in** (state x) and **not in** (state x).

Effect: The 'effect' is behaviour, normally an activity or an opaque behavior, executed during the transition from one state to another. If the transition is triggered, then first the exit behavior of the current (source) state is executed, then the transition effect is executed, and finally the entry behaviour of the target is executed. A state machine can contain transitions called internal transitions that do not effect a change in the state. An internal transition has the same source and destination and if triggered, simply executes the transition effect by contrast, an external transition with the same source and destination state –sometimes called a 'transition-to-self' trigger, execute the execution of that state's entry and exit behaviors as well as the transition effect.

Entry and exit points on state machine: For a simple region state machine entry- and exit-point pseudo states are similar to functions they are part of compound transition. Outgoing guards have to be evaluated before the compound transition will be taken. On the state machines, entry-point pseudo states can only have incoming transitions. Entry- and Exit-points are described by 'small circles that overlay the boundary of a state machine or composite state'. The entry-point symbol is hollow, whereas an exit-point symbol has a cross, rotated 45^0 from the vertical/horizontal axis [4].

State hierarchies: Just as state machines can have region so can states, and such states, are called **composite** or **hierarchical state**. This capability allows state machine to scale to represent arbitrarily complex state-based behaviours.

APPLICATIONS

CASE 1:

It is possible that events may trigger transitions at several levels in a state hierarchy and with the exception of concurrent regions, only one of the transitions can be taken at a moment. Let us consider the state machine shown in fig1. Here, priority is given to the transition whose source state is innermost in the state hierarchy. The machine has its initial state (state 1.1.1 and 1.2.1). The signal, *sig 1* is associated to the triggers of three transitions, each with guards based on the value of variable x

The followings explain the behavior of the state machine and its transition firing order based on the value of x (from -1 to 1) :

- X equal -1; transition t_1 will be triggered because it is the only transition with valid guard.
- X equals 0; transition t_2 will be triggered because although transition t_1 also has a valid guard, state 1.1.1 is the innermost of the two source states.
- X equals 1; here, both transition t_2 and t_3 will be triggered because both their guards are valid.

Therefore, before the transitions 1 to state 2 can be taken, any exit behaviour of state 1 must be executed.

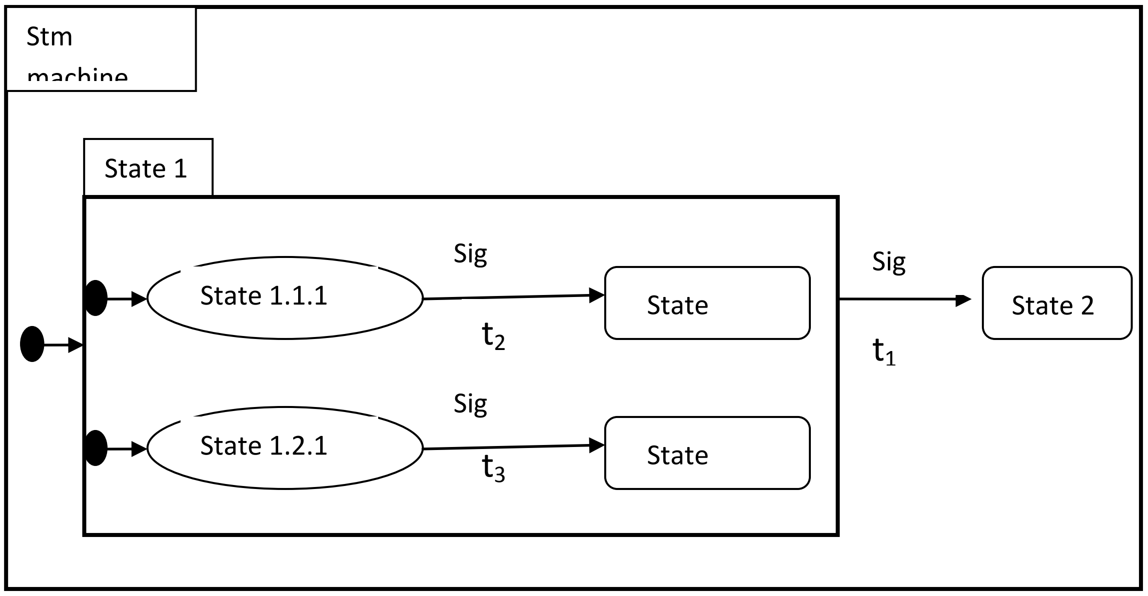

Fig 1Transition firing order

CASE 2 : Fig 2 shows a simple state machine for controlling a lamp with an unlatched button. It starts in state **off**, which has an entry behaviour that turns lamp off and a do activity that repeatedly polls an input line and places the value of the input into the variable button value. A change event, when (button value = 0), triggers a transition to state **on**, so as soon as the polled value changes to 1, the **off** state is excited and the do activity is terminated. On entry into the **on** state, the lamp is turned on and the state machine again is triggered by the change event when (button value = 1).

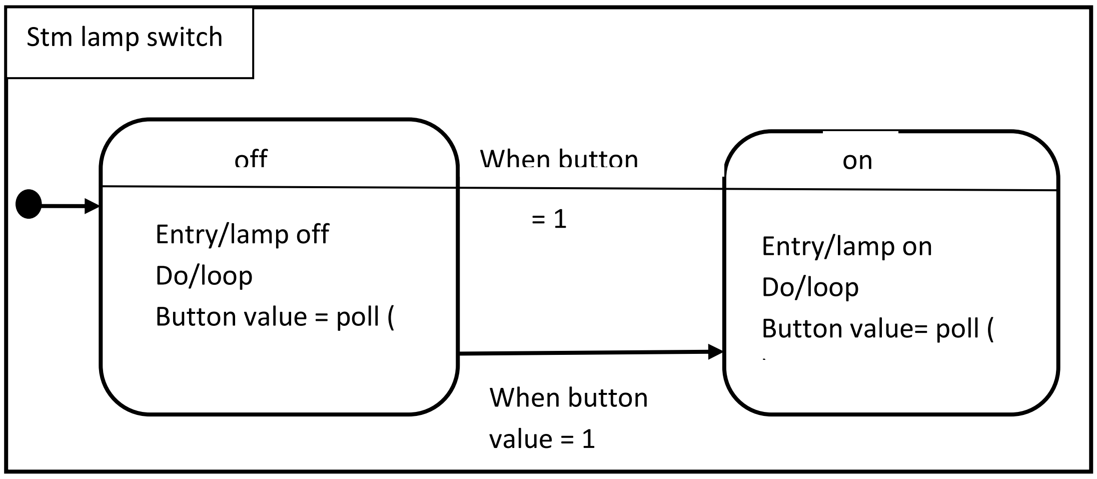

Fig 2 Stm lamp switch

CASE 3
The state machine for H_2O shown in fig 3 defines the transition between its solid, liquid and gas states. These represent discrete states of H_2O, while the value of its properties, such as temperature and pressure represent continuous state variables. Specific values for the temperature, plus other conditions (e.g. the withdrawal or addition of energy), define the expressions for the change events on the transition. So implicitly, the values of its state variables determine the discrete state of H_2O via the transition between them.
From fig 3, when temperature is at 100^oC & latent heat of evaporation is added, the water (H_2O) changes from liquid state to gaseous state. While temperature is still at 100^oC & latent heat of evaporation is removed, the H_2O changes from gaseous state back to liquid state. While it is still in liquid state but temperature is at 0^oC & latent heat of liquefaction removed, the H_2O changes from liquid state to solid state. While it is still in solid state and temperature value is still 0^oC but latent heat of liquefaction is added.

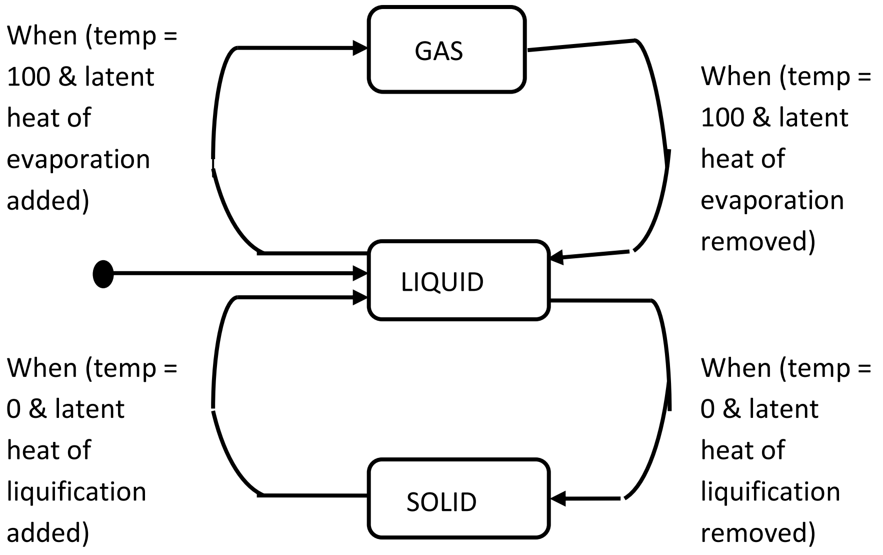
When (temp = 100 & latent heat of evaporation added)
GAS
When (temp = 100 & latent heat of evaporation removed)
LIQUID
When (temp = 0 & latent heat of liquification added)
When (temp = 0 & latent heat of liquification removed)
SOLID

Fig 3 machine for H2O

III. DISCUSSION

As can be seen, **case 1** expresses the semantic of hierarchical state decomposition and is equally designed to facilitate re-using of behavior. From fig 1, it has been demonstrated that instead of facing all aspects of a complex system at a time, it is imperative to ignore some parts of the system abstract away at a time. This mechanism does not just hide complexity in a system but also acts as a powerful mechanism of hierarchical event processing. The type of solution in **case 2** is a discrete machine driven by change event. It could be suitable for describing digital systems that execute continuously, monitoring inputs and writing outputs. Here, transition between states in the lamp switch state machine is triggered by a change to the value of a discrete property, button value. This is in contrast to the continuous state representation of a system in terms of continuous state variables (expressed as value property) in **case 3**.

IV. CONCLUSION :

State machines are used to describe behaviour of a block in terms of its state and transition. Change events are driven by the values of variables of the state machine or properties of its owning block. In addition to discrete systems change events can be used to describe the state of continuous systems, where transitions between the system's discrete states are triggered by changes in the values of other continuous properties.Generally, modeling and control of event-based behaviour using state machines is very important in event-driven programming because it makes the event handling explicit dependent on both the event-type and on the state of the system when correctly used. It can as well drastically cut down the number of execution paths through the codes, simplify the condition tested at each branding point, and simplify the switching between different modes of execution.Finally, this study should enable the reader to understand and appreciate how the concepts of state machines are applied in real life situation.

RECOMMENDATION : We recommend that other modeling constructs such as; Parametric, Use Cases, Interactions, Activities, Blocks etc can be employed to model event based behaviour.

REFERENCE

[1] **Hamon, G**. (2000). Denotation semantics for state flow. International conference on embedded software. Jerseey City.

[2] **Alur, R., Kanade, A., Ramesh, S., Shashidhar, K. C.,** (2008). Symbolic analysis for improving simulation coverage of simulik/state flow models. International conference on embedded software, Atlanta.

[3] **Samek, M.** (2003). 'Who moved my state?' C/C++ Users Journal, The embedded angle column.

[4] **Douglass, B. P.** (1999). Doing Hard Time Developing Real Time Systems with UML, objects, frameworks and patterns.

[5] **Tiwari, A**. (2002). Formal semantics and analysis methods for simulink state flow models.

[6] **Harel, D., Peliti, M.** (1998). Modeling reactive systems with state charts.

[7] **Samek, M**. (2008). Practical modeling language statecharts, 2nd edition. Event-driven programming for embedded system.

[8] **OMG** (2009). 'OMG Unified Modeling Language Superstucture Version 2.2'.