State Chart

Mitul Shah (CE – 104)
Sumit Vachani (CE – 122)

Abstract

This paper presents model used for generation of state chart in embedded system containing hardware and software. State Charts are basically used for visual representation of complex systems. State chart extends the conventional state transaction diagram with hierarchy, concurrency and communication. Thus by state chart we can divide large information in small chucks that can be understood easily by anyone as said truly "A picture is worth a thousand words". At last we have also included the illustration of how to draw state chart using a tool named ArgoUML, this tool also provides facility to draw other UML diagrams also. At after that we have defined some pros and cons of using state chart in Embedded Systems.

1. Introduction

The Unified Modeling Language (UML) state diagram is essentially a state diagram with standardized notation that can describe a lot of things, from computer programs to business processes. State charts were developed by David Harel, to add nesting and other features to flat state machines and were later added to UML and standardized. They are an excellent tool for modeling modules (or classes), sub-systems and interfaces that have many distinct states and complex transitions among them.

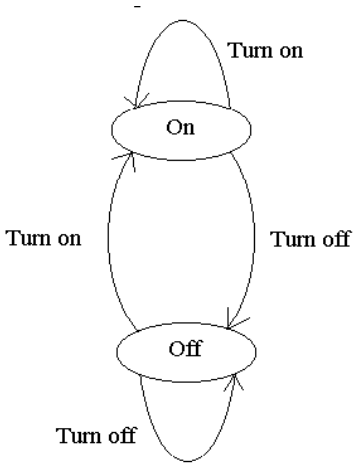


Figure 1: Simple State chart.

The question is why State Chart came into existence? Why the conventional state transaction diagrams were not enough to represent the complex data? The answer to both of these questions is the traditional diagrams were unable to show concurrent execution of system, they were unable to manage large amount of data, and if we try them with large data then it makes whole representation more ambiguous. Traditional systems were also unable to present communication between two states. The state charts are mainly for OO (Object Oriented) behavior. Another major benefit of using state chart is that it is language independent. It should also provide support the non standard IO devices (i.e. direct access to switches, display terminal, etc.). State charts are mainly used for reactive systems categorized by being event-driven, continuously having to react to external and internal stimuli. Some famous examples are telephones, automobiles, missiles, communication networks and many more.

Here a simplest kind of state chart is shown in figure 1. The chart shows the ON/OFF mechanism of any electric system. There are mainly two states On and Off. You can get into any of state by performing action provided on arrow. That is arrow shows the required event to reach to that state. You can also provide some condition such that that event will only occur if that condition is true.

You can draw state chart of any system which posses some particular states and at any time that system is in any of that states. In embedded systems the state charts are used in designing part. By design of whole system you can predict the complete behavior of the system. That is main purpose of charts.

There is also two subtypes of state chart

- AND – State machines: in which machine can be only in ONE state at any time.

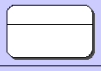
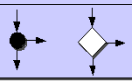
- OR – State machines: in which machine can in two diff states of chart at same time. This means we can represent concurrent system actions in OR state machines.

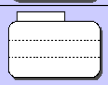
A state can be divided in 2 areas: the upper for its name and the lower for its actions. Actions are divided in:
Entry actions : Executed entering the state
Do actions : Executed inside the state
Exit actions : Executed leaving the state
Event actions : Executed due to an event (specified inside a transition)

2. Brief History

- Early 1980's: David Harel invents State charts in Israel, intended for specifying embedded software for fighter jets.
- At the same time: G´erard Berry develops the textual language Esterel in France.
- 1987: I-Logix is founded in the U.S. and releases the first commercial State charts tool, called Statemate.
- Mid 1990's: Charles Andr´e develops SyncCharts in France, a purely synchronous variant of State charts and a visual front–end for Esterel.
- 1998: MathWorks releases Stateflow for their Matlab/Simulink toolbox.
- 1999: Esterel Technologies is founded in France and releases the design suite Esterel Studio which integrates Esterel and SyncCharts ("Safe State Machines").
- 2001: Esterel Technologies acquires SCADE from Telelogic and integrates Safe State Machines into SCADE.

3. State Chart Notations

Description	UML symbol	Description	UML symbol
State		Pointing to the initial state	
Transitions		Indicating the final state	
Self transitions		Flow final	
Junction point		Choice point	

Description	UML symbol	Description	UML symbol
Composed state		Join	
State and concurrent substates		History	
Fork		Shallow history	

This two tables shows the notations that are being used in state chart. Some of them are described below.

I. Transactions

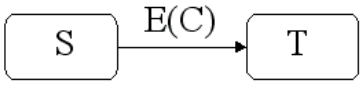


Figure 2: Simple transaction.

Here S and T are two different states while E stands for event and C for condition required for this transaction. If event E occurs in state S and condition C holds then make the transition to state T.

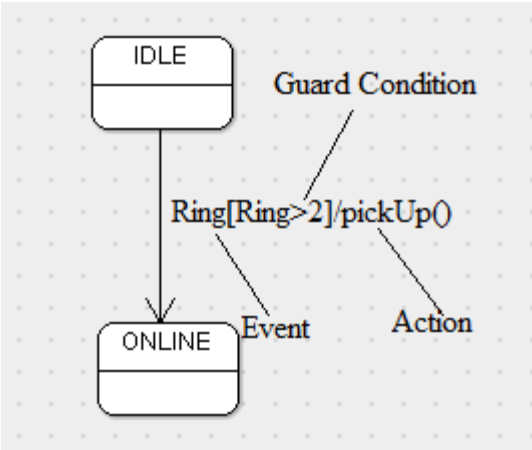


Figure 3: Transaction with event, guard ,action.

Graphically, UML shows states as boxes with rounded corners and transitions as arrows lines. The transitions are labeled with the ***event*** that causes the transition. A condition called ***guard can be indicated*** in square brackets followed by the ***action(s)*** that will be taken upon transition.

II. Hierarchy Representation

FSM are flat. They provide no facility for hierarchical model. Sometime details might be hidden in such states.

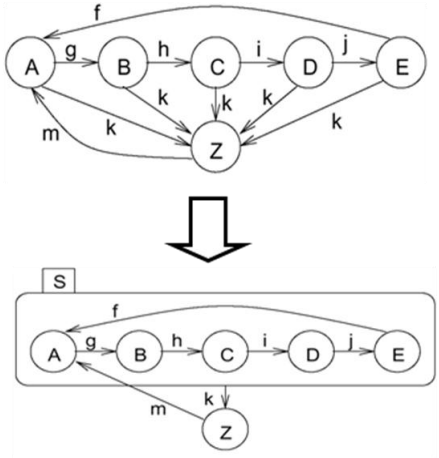


Figure 4: Representation of hierarchy

Figure 4 shows how hierarchical representation of simple system. Just by looking at figure we can judge that how much it's easy to understand a hierarchical model then a flat one. Here we have combined the five states A, B, C, D, E into one state S.

The different elements displayed in Figure 5 are:

- Current states of FSMs are also called **active** states.
- States which are not composed of other states are called **basic states.**

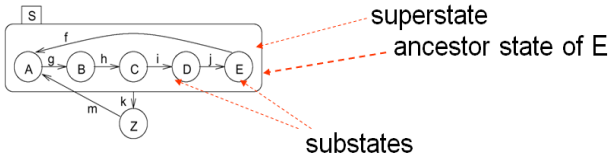


Figure 5: Elements of hierarchical statechart

- States containing other states are called **super-states**.
- For each basic state s, the super-states containing s are called **ancestor states**.
- Super-states S are called **OR-super-states**, if exactly one of the sub-states of S is active whenever S is active.

Here S is the main state by which we can manage all states belonging to that. Any arrow going into state S or moving away from it can be from any of the states in S.

To identify which state to enter when any incoming arrow is drawn to S we have two different options:

1. Use Default Mechanism
2. Use History Mechanism

You can also provide combination of both. Such that if it's first entry for that state it will use default state otherwise t will use the last used state (History).

III. Start and Stop states

The start states are represented by simple large dots. Where stop states are represented by encircled dots.

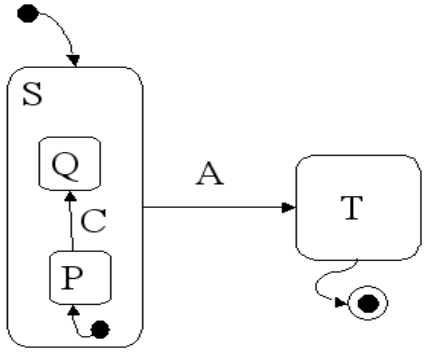


Figure 6: Start and Stop states

As we can see in the figure the S state is starting state and T represents the ending state. The default state for S is P. That is whenever any machine enters S it will first be in state P.

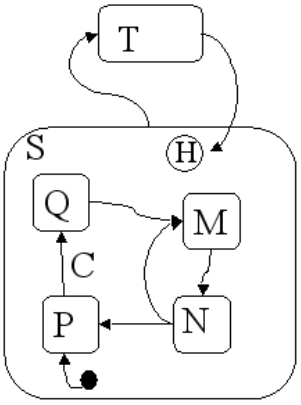


Figure 7: Start and stop states with history mechanism.

When entering a state that has a nested statechart, you may want to resume where you left off when you left the enclosing state. The encircled H as the entry point as below means to start in the state within that was exited.

Each level can have its own history mechanism. Each history variable would be initialized to the start state of the level.

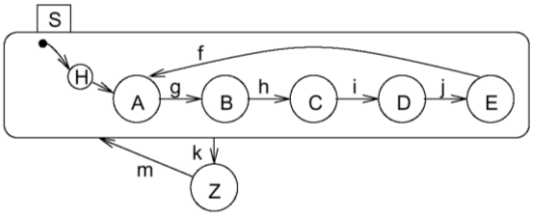


Figure 8: Combining history and default state mechanism.

IV. UML Junction Point

It is not mandatory that a junction point has transitions to provide every possible conditions to change state. In the example below there isn't a transition for 3<a<0. What does it mean? Simply that for 3<a<0 there will be not a transition to LEFT or RIGHT states and IDLE will remain the active state.

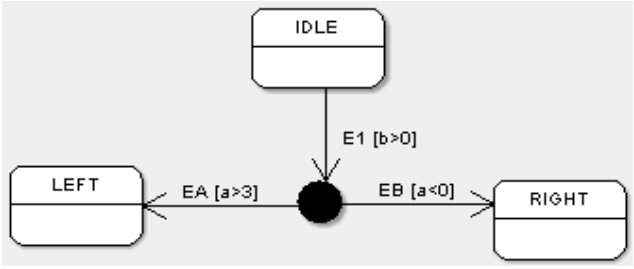


Figure 9: Showing UML junction

V. Timers

Since time needs to be modeled in embedded systems, timers need to be modeled. In StateCharts, special edges can be used for timeouts.

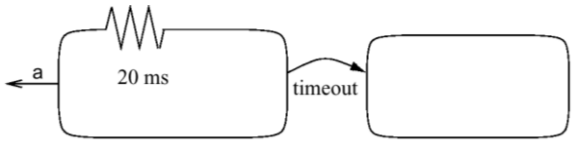


Figure 10: Representation of timers in state chart.

If event a does not happen while the system is in the left state for 20 ms, a timeout will take place.

4. How Statecharts works?

To begin understanding statecharts, it is best to start with the classic state diagram and then add the notions of hierarchy, concurrency, and events. The classic state diagram consists of two main constructs: states and transitions. In **Figure 11**, the state diagram describes a simple soda vending machine with five states and seven transitions to illustrate how the machine operates. The machine starts in the "idle" state and transitions to the "count coins" state when coins are inserted. The state diagram shows additional states and transitions when the machine waits for a selection, dispense a soda, and gives change.

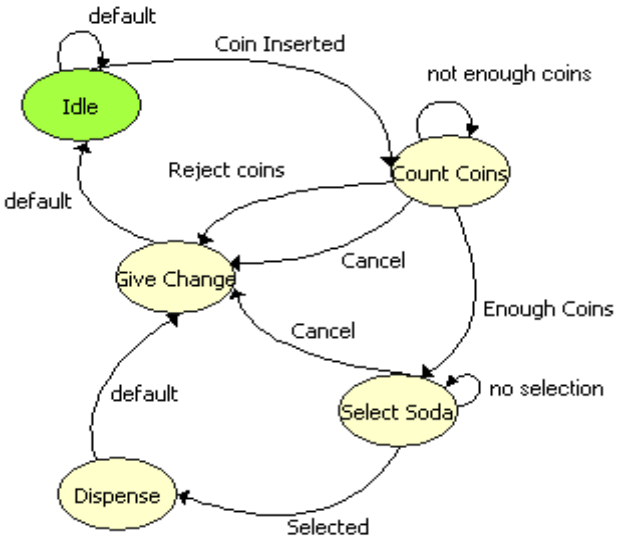


Figure 11: State Diagram Describing a Simple Soda Vending Machine

Figure 12 shows a statechart that describes the behavior of the same machine. Notice how the notion of hierarchy and events reduces the number of states and transitions. In the statechart, you can nest the "count coins" and "dispense" states within a superstate. You have to define only one transition (T3) from either of these two states to the "give change" state. You can configure the T3 transition to respond to three events: soda dispensed, change requested, or coins rejected. Additionally, you can eliminate the "select soda" state in the classic state diagram by introducing a "guard" condition to transition T2. Guard conditions must evaluate to "true" for the transition to occur. If the result of the guard condition is "false," the event is ignored and the transition does not take place.

At this point, you can expand the statechart to demonstrate the notion of concurrency by adding a temperature control element to the software within the vending machine. **Figure 13** shows how you can encapsulate the dispensing logic and the temperature control into an *and-state.* And-states describe a system that is simultaneously in two states that are independent of each other. The T7 transition shows how statecharts can define an exit that applies to both sub-statecharts.

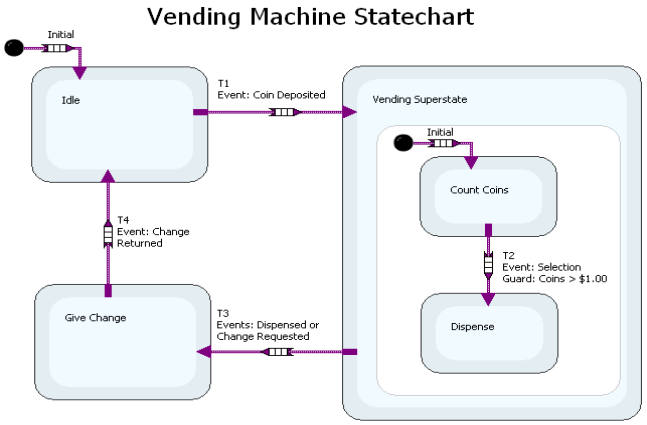


Figure 12: Statechart Describing a Simple Soda Vending Machine

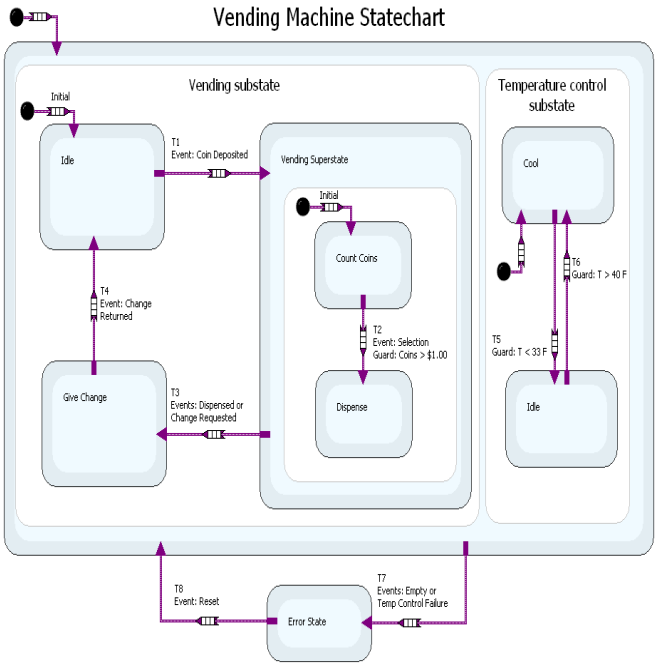


Figure 13: Encapsulating the Dispensing Logic and the Temperature Control into an And-State.

In addition to hierarchy and concurrency, statecharts have features that make them valuable for complex systems. Statecharts have a concept of history, allowing a superstate to "remember" which substate within it was previously active. For example, consider a superstate that describes a machine that pours a substance and then heats it. A halt event may pause the execution of the machine while it is pouring. When a resume event occurs, the machine remembers to resume pouring.

5. Tools

There are many tools available to draw state charts. Some of them are SCADE suit, Smartstate, Argo UML, StateMate. Here we have used a tool ArgoUML which is freeware available for downloading at http://argouml-downloads.tigris.org/argouml-0.30.2/.

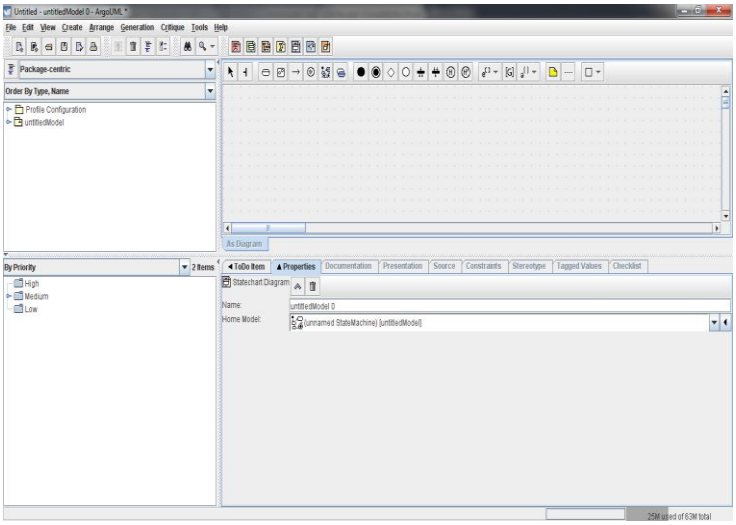


Figure 14: Screen shot of ArgoUML

Steps for using this software are described below:

1. Select New Statechart Diagram as shown in figure.

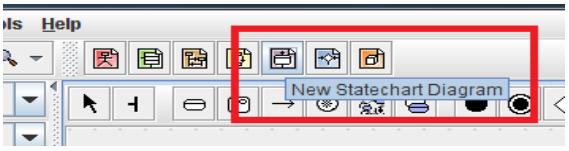


Figure 15: Selection of new state diagram in ArgoUML

2. After selecting the option, you will get a bar which displays different symbol used in state chart. You can add new state, new superstate, start/end symbol, etc.

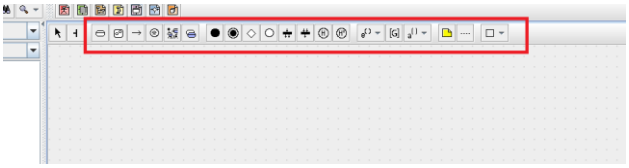


Figure 16: Different notations of state diagram

3. Suppose now you want to add a new superstate then simply select that option and click anywhere in workspace to add state.

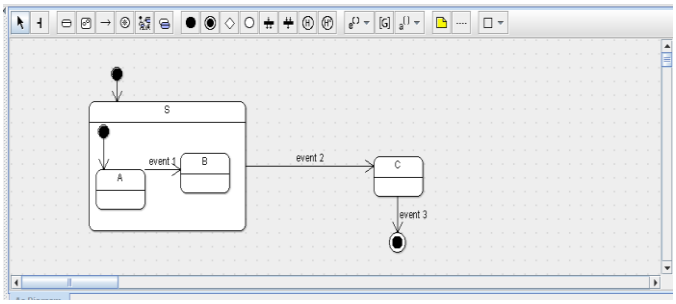


Figure 17: Example

4. Here we have shown simple statechart which has 3 simple states and 1 composite. State S is composite and contains two states A and B. The initial state is S as indicated by start symbol. And from C by performing event 3 system stops working.

5. This tool also provides facility to represent history and hierarchy mechanism.

6. You can also draw other different UML diagram. Like Use - Case, Class diagrams, Collaboration diagram, Activity diagram, etc.

6. Pros and Cons

Pros:

- Hierarchy allows arbitrary nesting of AND- and OR-super states.
- (StateMate-) Semantics defined in a follow-up paper to original paper.
- Large number of commercial simulation tools available (StateMate, StateFlow, BetterState, ...).
- Available "back-ends" translate StateCharts into C or VHDL, thus enabling software or hardware implementations.

Cons:

- Generated C programs frequently inefficient.
- Not useful for distributed applications.
- No program constructs.
- No description of non-functional behavior.
- No object-orientation.
- No description of structural hierarchy.

7. Conclusions

- Finite State machines
  - Simple
  - Many formal properties
  - Problem – too many states => Hierarchy
- StateChart
  - StateMate – Design environment for reactive systems
  - Behavioral modeling language for StateMate
  - Extends FSMs to better handle concurrency
  - Adds conditional statements, memory to FSM

So at the end we can conclude that statechart generally provides better representation of complex embedded system by reducing complexity. Even the developers of State-mate (tool for drawing statecharts) says that their customers save 30% of total cost by using that software.

8. References

1. www.wikipedia.com
2. http://argouml.tigris.org/
3. http://www.benmeadowcroft.com/
4. White paper of Department of Electrical and Computer Engineering, College of Engineering, Technology and Computer Science