Object – Oriented System Development Using UML, JAVA and Patterns

Professor Rajib Mall
Department of Computer Science and Engineering
Indian Institute of Technology, Kharagpur

Lecture No 21

Features of State Machine Model

Welcome to this session.

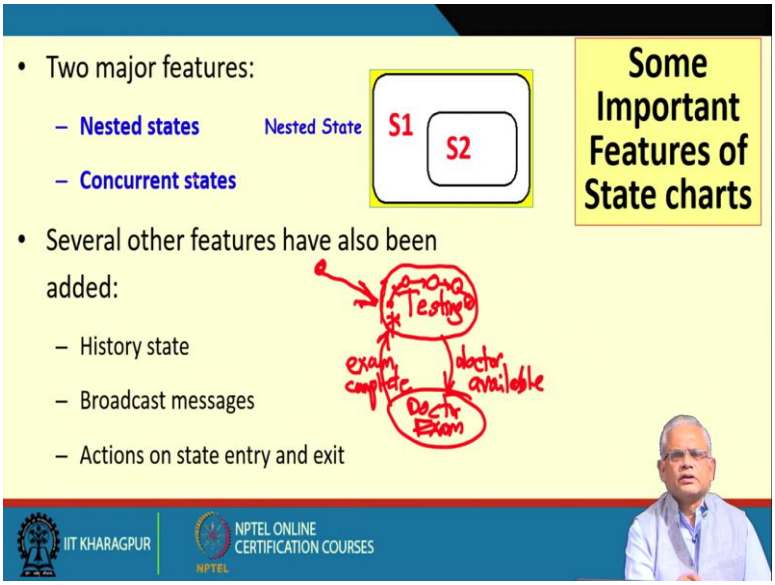
Over the last few sessions, we have been discussing about UML modeling of problems to be implemented using object-oriented languages. Initially, we started looking at the use case diagram, then we looked at the class diagram and for the last session, we have been looking at the state machine diagram.

The state machine diagram is much more powerful than the finite-state machine. The finite-state machine had the difficulty of state explosion and lack of support for representation of concurrent states. The same system might have concurrency or maybe we want to represent different subsystems of the system and these subsystems have different states. And there is a support for this in the state machine diagram used in UML. The objects which have significant state models, it is necessary to construct the state machine model for that. It becomes easy to generate code from the state machine diagram.

Based on the state machine diagram, we can test the system well and also the chances of mistakes while representing the behavior in the implementation gets substantially reduced and it becomes easy to understand the behavior of the system. In a single sentence, we can say that the state machine diagram is very advantageous to use for objects having significant state behavior.

Now, let’s proceed with the basic notions. So far, we had studied about the state machine diagram. And let us see how are the state machine diagram different from the traditional finite-state machine.

(Refer Slide Time: 2:43)



There are two major features that have been incorporated in the UML state machine diagram. One is the nested states, this feature helps overcome the state explosion problem of the finite-state machine formalism. We will see how the nested states are represented. Another one is the concurrent states.

The nested states, we will see that one state will have another state machine inside it or maybe multiple state machines inside it (in the above slide). So, at the top level, we might see just one state S1, but then as we explore further, we will see that S2 is another state machine model. We will see details of this as we proceed.

The concurrent states here, a single state represents two different states of the subsystems. They concurrently exist. Both of these features, the nested states and concurrent states, we call the states which are having this feature as the composite states. On the other hand, the states used in the finite state machine are the primitive states.

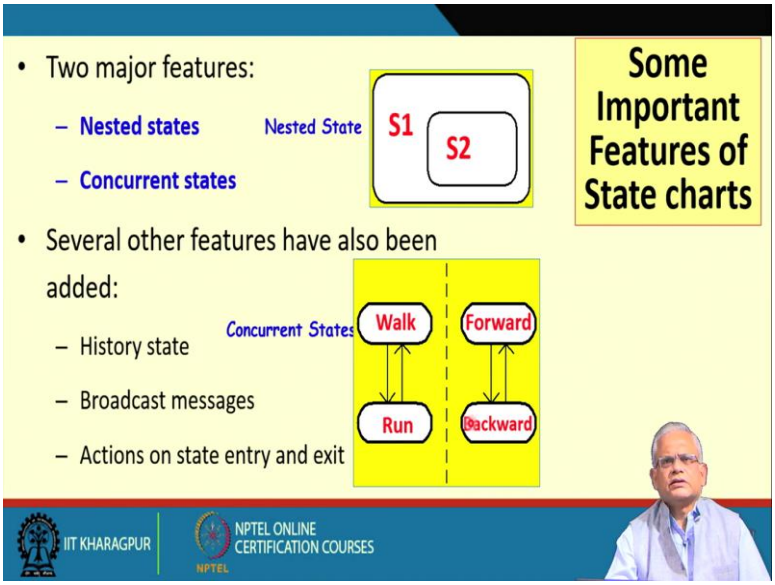
There are also several other features, the history state which enables an interruption of a state as the system is in a certain state, there can be an interruption. And the system can transit to another state temporarily to do something. And then it not only returns to the present state, but also it continues from the work that was done. Let just try to give an example of this, even though we are not going to see these three things, history state, broadcast messages, action on state entry and exit. All these we are not going to see in detail in this course, but we will just give an overview. Whereas the nested state and concurrent state, these two we will see in some detail, and also are going to use this in modeling systems.

As an example of the history state, let just consider a patient who is admitted in hospital. He is taken for testing. And the test there are many procedures, he gets the X-ray, then he goes for ECG and so on. There are a set of steps inside the test procedure in the hospital. From one test to another, he proceeds. And when the patient is in the testing state, various tests are being conducted. The specialist doctor has come and the patient is taken out from the testing state and returns is reported to the diagnosis test by the doctor and returns as the doctor completes his examination.

We represent that with the help of a '*'. In the above slide a hand drawing shown for this example. Let say the patient in the testing state, various tests are being conducted. From the pseudo state (filled round circle) the patient reports goes to Testing state. Another one is the doctor examination state and the patient goes to the doctor examination. The event that makes this transition is doctor available. And then the doctor examination is complete, the patient resumes testing not from the first step but the step at which he had been taken out for the doctor's examination. The Testing state here is the history state and we represent the history state with a '*' notation. This '*' notation indicates the history state in the sense that the history at which the testing was interrupted is remembered and as it resumed it continues from that point. We also need to label every transition with the event that causes the transition. Here, the transition Doctor available make transition from Testing state to Doctor Exam and the transition 'examination complete' lead to Doctor Exam to Testing State.

But we are not going to use the history state because the simple problems that we will solve, we don't have scope for using these other features: history state broadcast messages, accents and state entry and exit. But we will use nested states and concurrent states, these are two very important features of state charts.

(Refer Slide Time: 9:57)



Now, let us look at these two features, the nested states and the concurrent state charts.

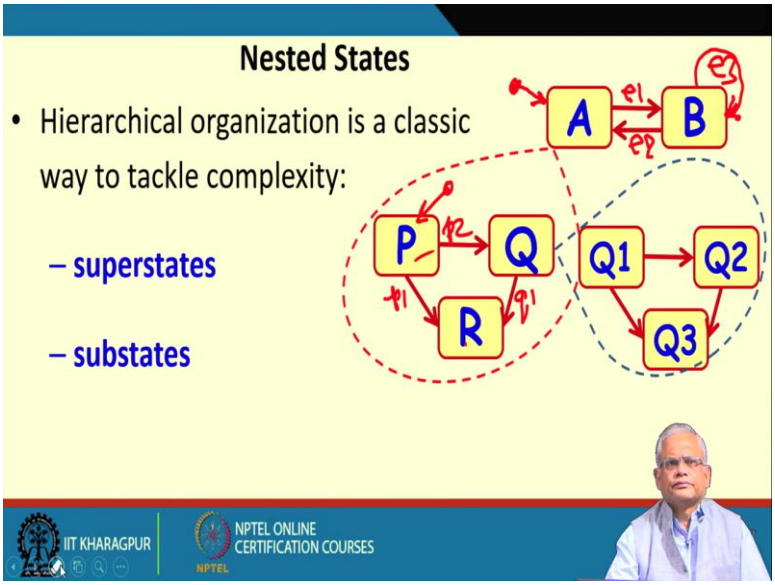
The concurrent state, these are two subsystems or two features. Let say a robot (in the above slide) can walk and run and we can annotate the event that causes it. If the Run button on the remote is pressed, when the robot is walking, the robot transits to the running state, and if the Walk button is pressed the robot goes back to the walking state.

But then, the same time, we have another two buttons, the backward and forward. Let say, to start with, the robot is walking in the forward direction. And if we press the backward button, then the robot is walking backwards. And if the forward button is pressed then the robot walks forward again. And therefore, the state of the robot is not only walking or running but also Forward and Backward at the same time. The robot may be walking backward or maybe walking forward or maybe running forward or maybe running backwards.

So, all possible combinations between these states are possible. And we call this in state chart is AND composite state because not only that it embeds two state machines, but also it is simultaneously present one state and another state and therefore, it is called as AND state: one from one state machine and one at another state machine at the same time.

We represent that in the form of a dotted line here that indicates the AND state. On the other hand, we have the OR state where it is present in one of the state machine only and therefore, if we have two state machine there it is present in one of the state machine only and that we call as OR state, but if we have this dotted line, then we call it as the AND state.

(Refer Slide Time: 13:08)



In the nested state, we represent the behavior of the state machine using a hierarchical diagram. We had said that in the FSM, one of the big problem was state explosion. As the number of aspects of the system increases, the number of states increase exponentially. If we have 10 different aspects, let us say there are two possibilities, then we have $2^{10} = 1024$ states. Which is a large number of states, becomes extremely difficult to understand the behavior, but using the nested state and the composite and the end state it becomes easy to represent and easy to understand.

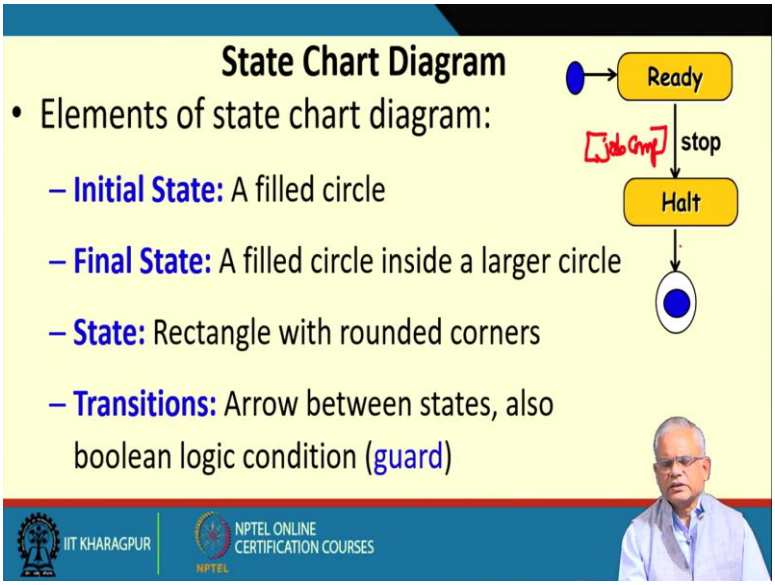
Now, let us look at the nested state. Using the nested state, we might represent the behavior of a system is having two states. Let us say the states are A and B (in the above slide). And we can also draw the start pseudo state. It starts at A and then it can transit to B. Let us say the event causing are e1 and e2.

The system starts at A and as the event e1 occurs, goes to the B state and there may be other events like e3 which might be occurring internally. On e3 state B remain in B. Using the nested state, the top-level behavior becomes very easy to understand that there are only two states A and B and we understand what's A and B. And then maybe we can say that A is startup and B is operational. The machine starts and is operational. The startup complete goes to operational and the stop switch is pressed, it goes to shutdown state. At the top level, it becomes very easy to understand the behavior, but then A and B will contain further state machines. In a case tool, if we double click on A, it will show another state machine or we double click on B, it will show another state machine. We call A and B as superstates. If we double click A, we will see a substate of A.

We can find that A embeds another state machine P, Q, and R which are the substates. And we might have a pseudo state in the the substates. As to start with, as it goes to A actually it will be in P. And then based on the events, P1, P2, and Q1 transition occurs. So, in this representation, to start with the system is in A state, but if we look deeper, it is actually in P state, because the pseudo transition present in substates. And then if the event P1 occurs, it goes to the R state, if P2 occurs, it goes to Q state and from Q state if the Q1 event occurs, it goes to R state. Initially, we represent it is a very simple diagram, but if we we looked at deep we can observe, there is another state machine embedded inside the state. So, here A is the superstate, P, Q, and R are the substates. And maybe, we might have some of these as again, composite states, which might embed further state machines. For example, maybe Q has further substates, Q1 Q2 Q3 and then Q3 might have further substates and so on. This kind of representation helps us to understand the model easily instead of representing thousand substate in one page.

So, to summarize this, we say that at the top level there are only two states and A has three substates. And then as you understand the behavior of this well, we look at the behavior in the Q state and we find that actually, the Q behavior has another state machine diagram and so on. The nested states help us to represent a system in a very understandable way and it is a recommended practice to wherever possible to represent the model using the nested states.

(Refer Slide Time: 20:02)



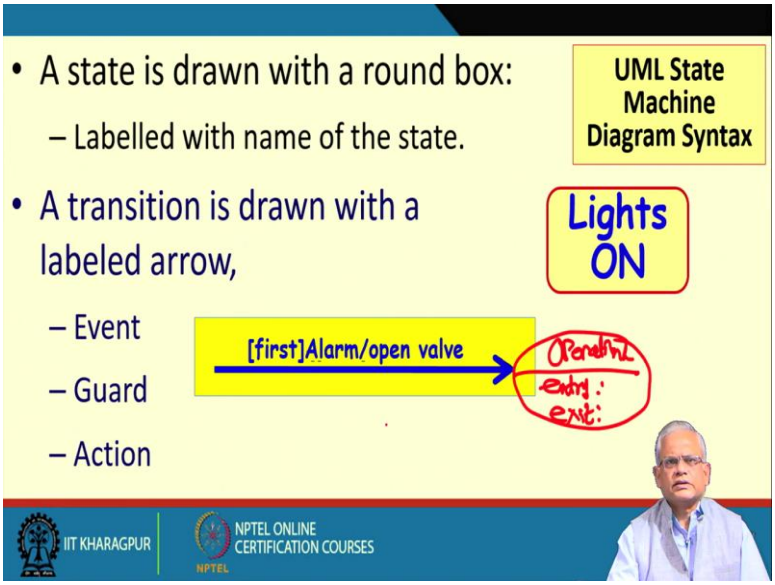
Now, let's look at the features of a state chart diagram.

In UML, it is called as a state machine diagram. We have the initial state, which is a pseudo state, in the sense that it is represented by a filled circle (in the above slide, first filled blue circle). It is a pseudo state, because the system is in a transient state. When system in pseudo state it transits to start state without any event. And therefore, we call this as the pseudo state. Pseudo state is a state where the system does not spend time and on a pseudo transition from pseudo state the system enters the Ready state (as shown in the above slide).

A state is represented with a rectangle with a rounded corner. The transitions are open arrows. We annotate the event that causes the transition in transition arrow. For example, here from ready state to halt state that is the stop switch event is occurred. We might also have a guard associated with the transition (within [] bracket).

We might say that the job complete, that is the guard and then the stop switch is pressed, it halts. If the job is not complete and you press the stop switch, nothing happens, it continues to be ready and doing whatever that may be represented inside this super state. The state through which it is doing work may be represented. But only after the job that it is doing is complete and you press the stop switch, then it goes to the halt state. Therefore, the guard is a condition, if that is fulfilled or turns out to be true and then the event occurs, then the transition is taken.

(Refer Slide Time: 23:19)



The state is drawn with a round box. Very easy to draw with a case tool, but a bit difficult to draw by hand, but then we will get used to it. It is labeled with the name of the state. The transition is drawn with a labeled arrow between the states. The label contains an event and a guard.

The label can contain an event, which causes the transition. A guard, which is the condition which must be satisfied and the event occurs then the transition is taken. Just the event occurring does not cause the transition, the guard also has to evaluate to true. And also, we can represent an action that takes place when transits from one state to other. This is the state representation (in the above slide, Lights ON state), rounded rectangle and the name of the state Lights ON. On a transition between states (we use an open arrow here) and then on this, we write, the guard, the guard is written between two ([]) brackets. If the event alarm occurs and this is the first alarm, then we open the valve and then go to another state. So, open valve is action here. So, the action can occur even when the transition is occurring between the state. Before the other state is entered, the action would have completed. On the first alarm, the transition is taken and the valve is opened and it goes to another state.

So, that is the representation in UML. So, we will use this guard, the event and then the action that takes place as the system transits from one state to another, and before it enters the other state, this action is need to complete. But then the actions can also occur, some different actions may occur within the state, several actions may be taking place within the state, on entry, some action may occur before it exits, some other action may occur inside the state (as shown in the above slide).

If we compare this with Mealy and Moore machines, we distinguish between the Mealy and Moore machines based on where the action occurs, does the action occur on the transition or in the state. But if we look at this UML state machine diagram, the action can occur both on the transition and also in the state and therefore, it has features of both Mealy and Moore machines.

It is a much more powerful mechanism. As I was saying, it can model many things which simple FSMs cannot. And here, we might represent the entry and the exit that on entry to a state, we write the state is, let say operational state. And in that we write entry, it does something, we write the name of the action. Similarly, before it exits, it does something. But then in our examples, we are not going to use the entry and exit states.

We are just going to use the states, the transitions with guards, events and action.

We are almost at the end of this lecture. We will stop here and continue from the next lecture.

Thank you.