SE 1: Software Requirements Specification and Analysis

Lecture 6: UML State Diagrams

Nancy Day, Davor Svetinović


uw.cs.cs445
Today’s Agenda

- UML State Diagrams
  - Basics (continued from last lecture)
  - Hierarchy
  - History
  - Concurrency
  - State Diagram Semantics Exercise

We will continue with the stopwatch example.
Composite States

Basic states do not contain other states.

There are two kinds of composite states, which are states that contain other states:

- Simple (also called “hierarchical” and “OR-states”)
- Concurrent (also called “orthogonal” and “AND-states”)

A composite state combines states and transitions that work together towards a common goal.
Hierarchy
Hierarchy

- If a transition leaves a composite state, this transition applies to all the substates (sometimes called “submachines”). The substates “inherit” the transitions of the superstate.

- If a transition ends at a composite state, the transition is continued by default arrows indicating the initial state of the composite state.

- There should be a default arrow (pseudostate designating initial state) at every level in the hierarchy.

- Hierarchy can be used to abbreviate a flat (i.e., no hierarchy; contains basic states only) state machine. One transition leaving a superstate can represent many transitions in a flat state machine.
History
History

History is a pseudostate whose meaning is to designate the immediate substate at this level in the hierarchy that the system was in when the immediate parent state was last exited.

A history pseudostate can be the destination state of a transition or a default arrow.

A transition leaving a history state indicates what state to enter if the system has never been in this superstate before. While not always necessary, always having a transition leaving a history state is a good idea.

Usually lines entering a history state and leaving to a history state are not labelled.
Deep History

H*

Having a deep history pseudostate as the destination of a transition or a default arrow means that at all levels in the hierarchy below this one, the system should enter the substate that it was last in when that state was exited (i.e., apply history at all levels in the hierarchy below this one).

Notes:

- History and deep history states are pseudostates – no time is spent in them; they are just the continuation of a transition. (Although they are often called states.)

- Don’t use “H” as a state name!
Priority?

UML is not clear on whether a transition leaving a state lower in the hierarchy has priority over a transition leaving a state higher in the hierarchy. This can occur if both transitions are triggered by the same event or by two different events that occur at the same time.
**Priority?**

**Suggestion:** Use the semantics that a transition leaving a state higher in the hierarchy has priority over a transition lower in the hierarchy. This semantics means transitions leaving states higher in the hierarchy are similar to interrupts.

Note: If you use a different semantics than the suggestion above, note it in the “conventions” section of your SRS.

Another alternative is to have no priority, in which case either transition can be taken and your system is non-deterministic (i.e., either transition represents a possible behaviour of the system). The designer will then choose which behaviour is the one that will be implemented.
Single Input Assumption

Priority makes a specification more deterministic, but there can still be multiple transitions exiting the same state on the same event or on events that occur at the same time.

Another way to reduce non-determinism is to adopt the single input assumption, which means that you assume only one event can occur at a time.

The result of this assumption is that two transitions that leave the same state triggered by different events can never both be enabled at the same time.

If you make this assumption in your SRS, you must note it in the “conventions” section of your SRS.
Termination

ti is the name of the transition for reference purposes. It is not an event.

Final states do not need to be labelled.
Termination

A final state represents the end of computation within a composite state. (Recall that a final state is a real state.)

- A transition leaving a basic state that has no event or condition in its label is always enabled.

- If a composite state has a final state, a transition leaving a composite state that has no event or condition in its label is only enabled when the state is in its final state.

- Transitions based on events and/or conditions are enabled from any state within a composite state.
Concurrency

From: STATEMATE manual
Concurrency

- Substates within a concurrent state execute in parallel. Each has its own thread of control.

- The system must be in a basic state in each substate of a concurrent state at the same time.

- A transition in each substate of a concurrent state can be taken at the same time.

- History and deep history can be used in substates of a concurrent state.

- Transitions may leave one component of a concurrent state to a destination outside the concurrent state. All substates of the concurrent states are exited.

- A transition may enter one substate of a concurrent state. In this case, the default states of all other substates of the concurrent states.
Termination

- Lab1: lab done → Lab2: lab done
- Term Project: project done
- Final Exam: pass
- Incomplete

- Passed
- Failed
Termination

- Each substate of a concurrent state can have a final state.
- Execution of the concurrent state is considered finished when all its substates are in their final states.
- If a concurrent state has final states in its substates, a transition leaving a concurrent state that has no event or condition in its label is only enabled when all substates of the concurrent state are in their final states.
- Transitions based on events and/or conditions are enabled from any state within a concurrent state.
- The semantics if some substates have final states and other substates do not is not defined – probably not a good idea!
Termination

To terminate the behaviour of all substates of a concurrent states, use the terminate state.

![State diagram](image-url)
Coordination

Sometimes concurrent components are not completely independent and need to coordinate.

Ways to coordinate:

- Both taking transitions on the same event.
- Both components can check on the value of the same variables
  - `inState(x)`
    - a condition that is true if the system is in state `x`
    - `x` can be a basic state or a composite state

Watch out for race conditions! Usually, it’s not a good idea to have multiple concurrent components modifying the same variables.
Coordination

\[ A \]

\[ \text{eval} / x = \text{true} \]

\[ [x] \]
Drawing Conventions

The details of a composite state can be shown on a separate diagram. Use the **composition icon** in the state:

entry pseudostate

exit pseudostate
Validation

Recall one aspect of validating a state diagram: Given the list of possible events, for each state, consider whether each event $e$ is possible in each state $X$.

Now we have to consider hierarchy in checking the above property.

**Suggestion:** Carry out this validation on the basic states, but consider all transitions in the hierarchy.
Determinism

Some people say that requirements specifications should be deterministic.

If you want your specification to be deterministic, you must ensure that it is never possible for two transitions to be enabled at the same time.

Considering transitions at all levels in the hierarchy, watch for:

- Two transitions leaving a state triggered by different events (Are you making the single input assumption?)
- Two transitions leaving a state both triggered by the same event. (Are there conditions? Is there a priority?)
Good Style

- The best state diagram is the one that is the clearest. What does this mean?
  - Fewer transitions are better.
    - Use hierarchy to reduce the number of transitions
    - Don’t overspecify! If an event is not relevant leaving a state, don’t have a transition based on that event. (Occasionally this is alright because of the clarity provided by hierarchy.)
  - Use history and deep history
  - Use concurrency to recognize orthogonal aspects of the problem.

You will be marked on style in your SRS and on exams!
Summary

- UML State Diagrams
  - Basics (continued from last lecture)
  - Hierarchy
  - History
  - Concurrency
  - State Diagram Semantics Exercise

Next Lecture: UML State Diagrams III and using UML state diagrams to represent system state diagrams

Reading: same as last class