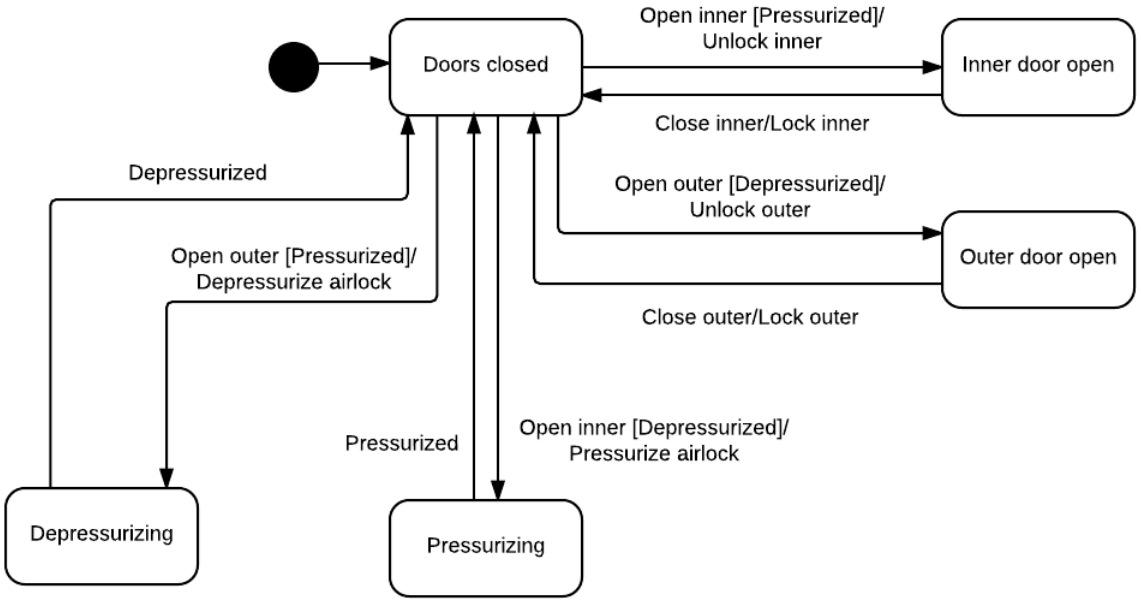
State-based Behavior I



SWEN-261
Introduction to Software Engineering

Department of Software Engineering
Rochester Institute of Technology

Software Engineering
Rochester Institute of Technology

A large part of software behavior is dependent on the state of the system, i.e. state-based.

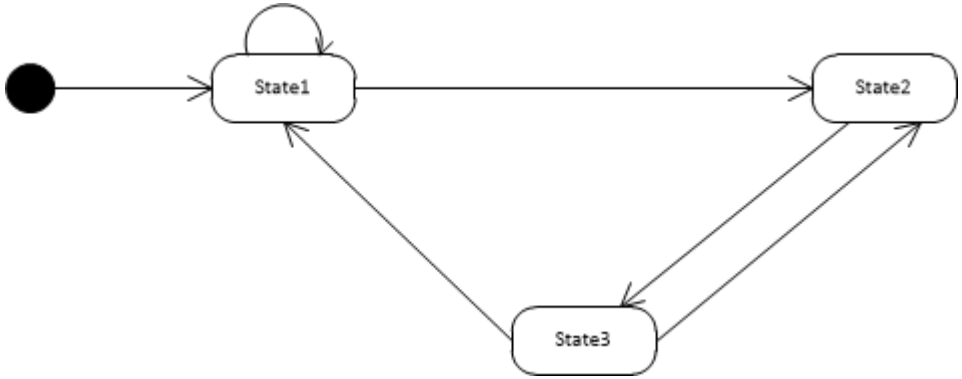
- A finite-state machine is a notational mechanism for capturing this state-based behavior.
  - ***The UML diagram is called a statechart.***
- Explicitly defining this state-based behavior provides a common specification for the team.
- Allowing the state behavior to evolve implicitly creates a situation where every team member may not have the same model of the behavior.

A statechart can define behavior in multiple areas in your application.

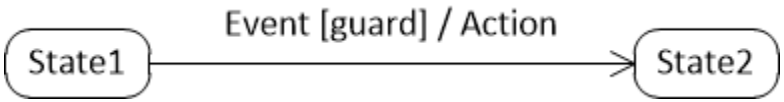
- Defining the operation of an interface, such as, the web application interface.
  - ***For an after class exercise, you will create a statechart that describes the sample webapp's web application interface.***
- Specifying the behavior for a single object.
  - ***Later in the course, we will come back to statecharts and discuss how to implement the state-based behavior explicitly.***

A statechart identifies the recognizable conditions that a system can be in over intervals of time.

- The system can exist in only one state at a time.
  - ***You typically want a deterministic state machine to define behavior of your software systems.***
- The system exists in a state for a period of time.
- A solid ball specifies the starting point.

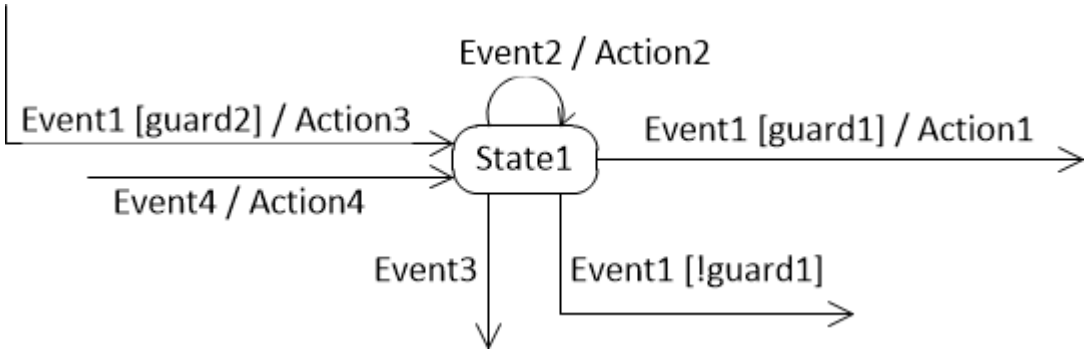


Transitions provide the mechanism for the system to move from one state to another.



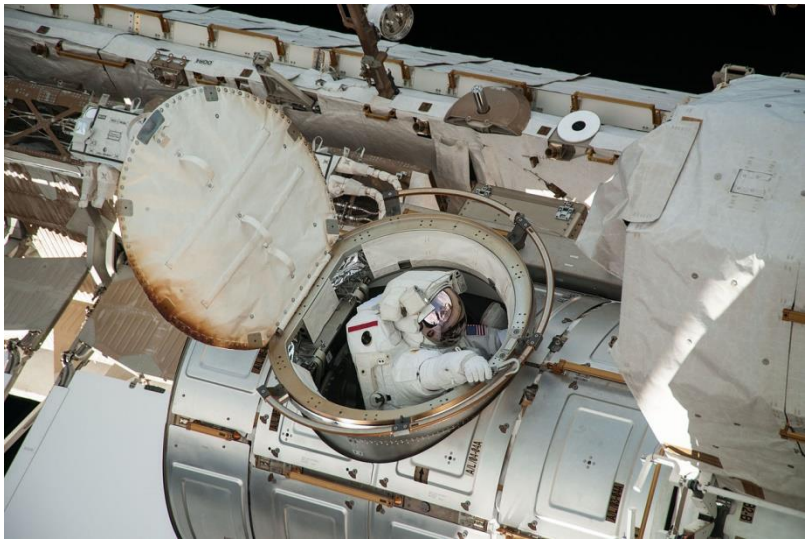
- The *event* triggers the transition to be taken.
  - ***The event could be calling a method, receipt of a signal, end of a time period, or end of an activity.***
- The *guard* is a Boolean condition that must be true for the transition to be triggered.
- When the transition is triggered, the *action* list is executed.

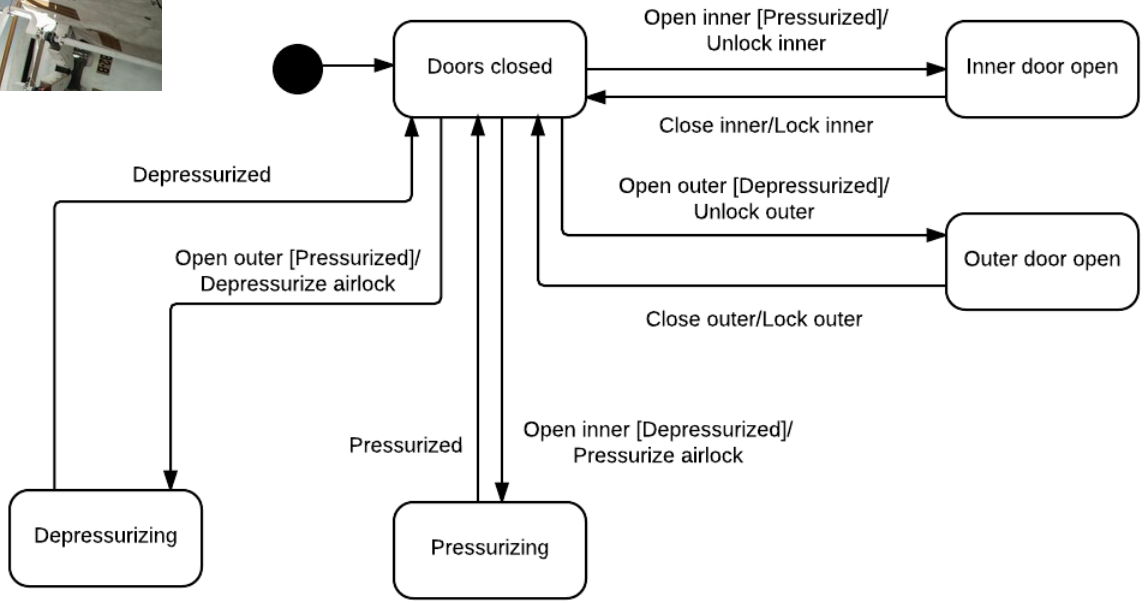
Transitions provide the mechanism for the system to move from one state to another.



- A transition executes instantaneously relative to the time spent in a state.
- A transition can return the system to the same state that it was in when the transition was taken.
  - ***Any actions would be executed before returning to the state just left.***
- There can be any number of transitions entering or leaving a state.

Here is a statechart for control of an airlock.





Follow these guidelines when creating your statecharts.

- Pick meaningful state, event, and guard names.
- Always specify a starting point without a guard on the transition to the initial state.
- Guards on multiple transitions from a state with the same trigger event should be mutually exclusive.
- Evaluating the guard should have no side effects, and the guard cannot use side effects of actions on a transition it is guarding.

Use a statechart to get a shared understanding of the state-based behavior of a system.

- Even if you do not explicitly implement the states, you can more clearly capture the system behavior.
- There are frameworks that provide an implementation of state-based behavior, such as StatefulJ FSM (https://github.com/statefulj) or squirrel-foundation (https://github.com/hekailiang/squirrel)
- The full statechart notation has even richer semantics defined (entry/exit actions, composite states, orthogonal/concurrent states).