# Elements of state diagram graphical notation. Features of object modeling in the state diagram.

## Introduction

A state diagram consists of states, transitions, events, and activities. You use state diagrams to illustrate the dynamic view of a system. They are especially important in modeling the behavior of an interface, class, or collaboration. State diagrams emphasize the event-ordered behavior of an object, which is especially useful in modeling reactive systems.

You use state machines to model the behavior of any modeling element, although, most commonly, that will be a class, a use case, or an entire system which focuses on the event-ordered behavior of an object, which is especially useful in modeling reactive systems.

## Key Concepts of a State Machine

A **state machine** is a behavior that specifies the sequences of states an object goes through during its lifetime in response to events, together with its responses to those events.

A **state** is a condition or situation during the life of an object during which it satisfies some condition, performs some activity, or waits for some event.

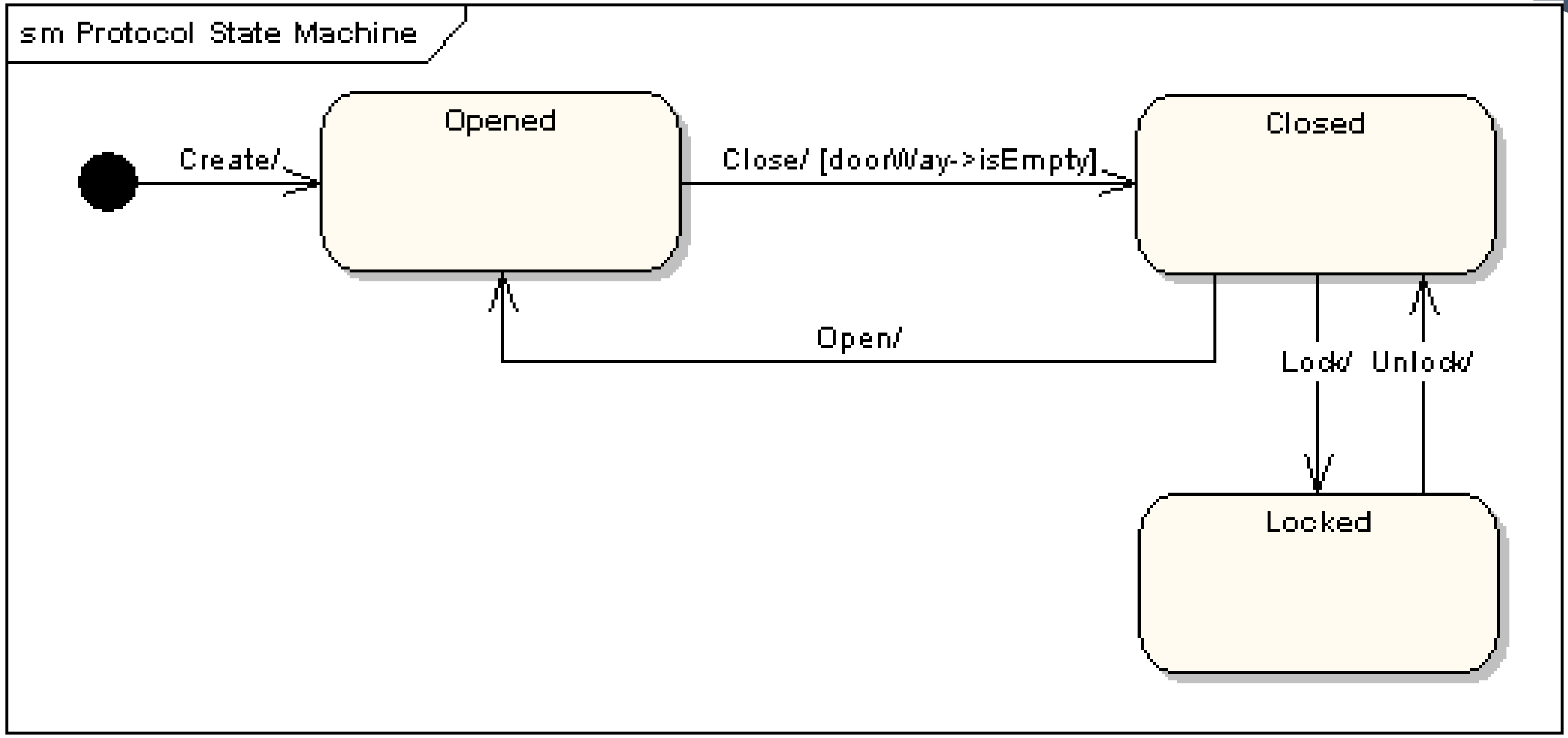
An **event** is the specification of a significant occurrence that has a location in time and space. In the context of state machines, an event is an occurrence of a stimulus that can trigger a state transition.

# Key Concepts of a State Machine

A **guard condition** is evaluated after the trigger event for the transition occurs. It is possible to have multiple transitions from the same source state and with the same event trigger, as long as the guard conditions don’t overlap. A guard condition is evaluated just once for the transition at the time the event occurs. The boolean expression may reference the state of the object.

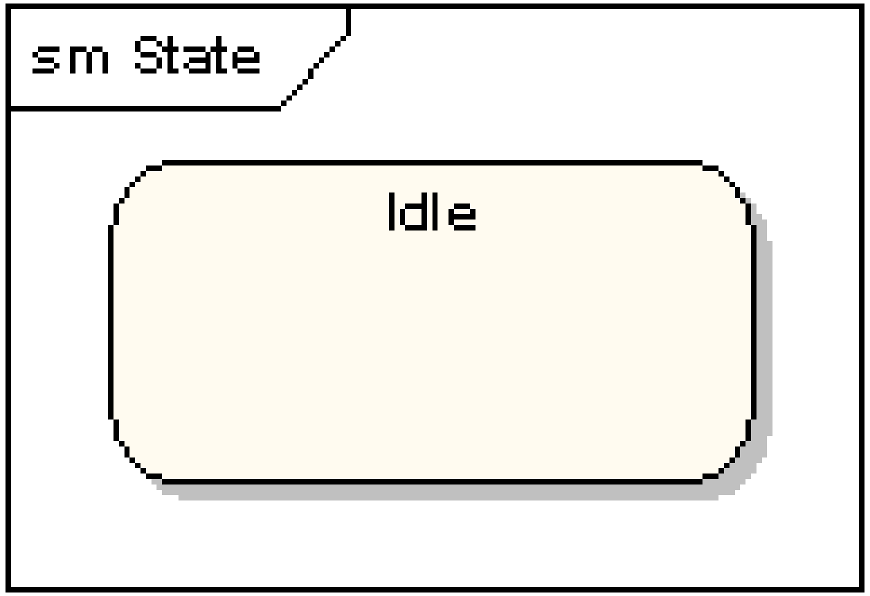
A **transition** is a relationship between two states indicating that an object in the first state will perform certain actions and enter the second state when a specified event occurs and specified conditions are satisfied. Activity is an ongoing non-atomic execution within a state machine.

An **action** is an executable atomic computation that results in a change in the state of the model or the return of a value.



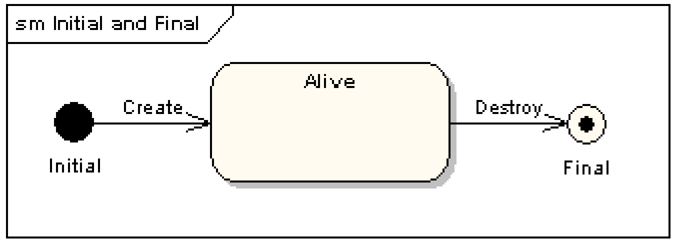
**The door can be in one of three states: "Opened", "Closed" or "Locked". It can respond to the events Open, Close, Lock and Unlock. Notice that not all events are valid in all states; for example, if a door is opened, you cannot lock it until you close it. Also notice that a state transition can have a guard condition attached: if the door is Opened, it can only respond to the Close event if the condition doorWay->isEmpty is fulfilled. The syntax and conventions used in state machine diagrams will be discussed in full in the following sections.**

# States



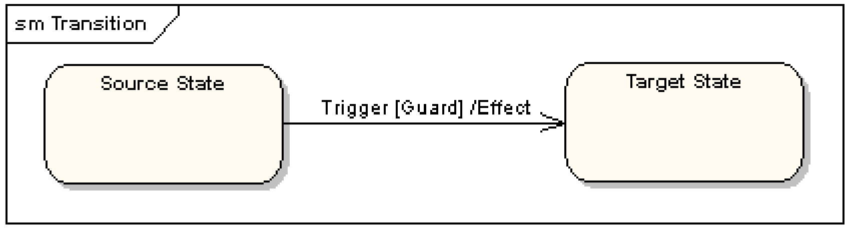
A state is denoted by a round-cornered rectangle with the name of the state written inside it.

# Initial and Final States



The initial state is denoted by a filled black circle and may be labeled with a name. The final state is denoted by a circle with a dot inside and may also be labeled with a name

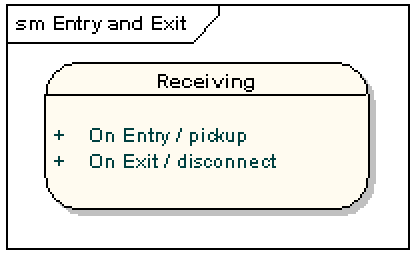
## Transitions



Transitions from one state to the next are denoted by lines with arrowheads. A transition may have a trigger, a guard and an effect, as below.

"Trigger" is the cause of the transition, which could be a signal, an event, a change in some condition, or the passage of time. "Guard" is a condition which must be true in order for the trigger to cause the transition. "Effect" is an action which will be invoked directly on the object that owns the state machine as a result of the transition.

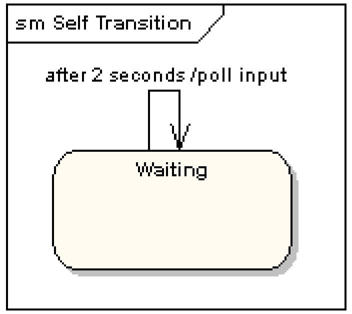
# State Actions



In the transition example above, an effect was associated with the transition. If the target state had many transitions arriving at it, and each transition had the same effect associated with it, it would be better to associate the effect with the target state rather than the transitions. This can be done by defining an entry action for the state. The diagram below shows a state with an entry action and an exit action.

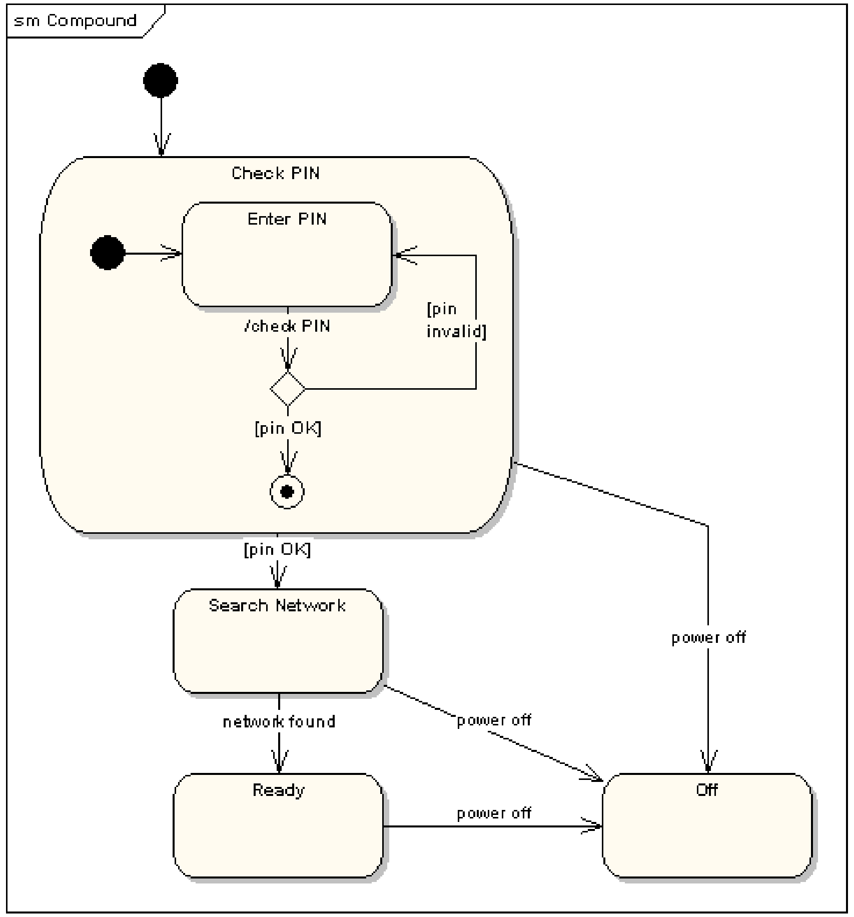
It is also possible to define actions that occur on events, or actions that always occur. It is possible to define any number of actions of each type.

## Self-Transitions



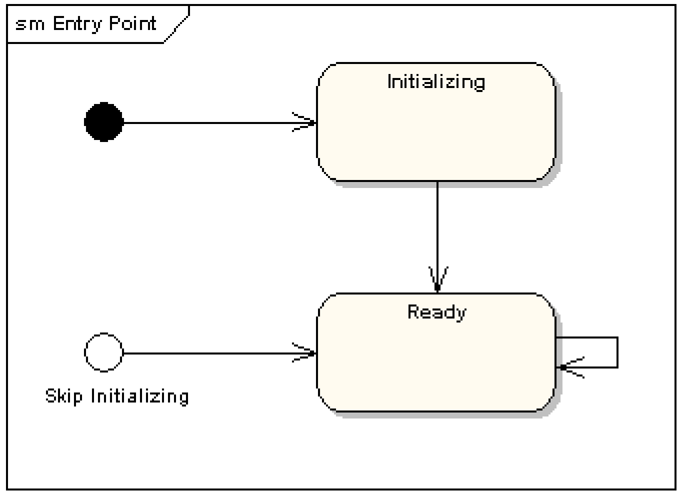
A state can have a transition that returns to itself, as in the following diagram. This is most useful when an effect is associated with the transition.

# Compound States



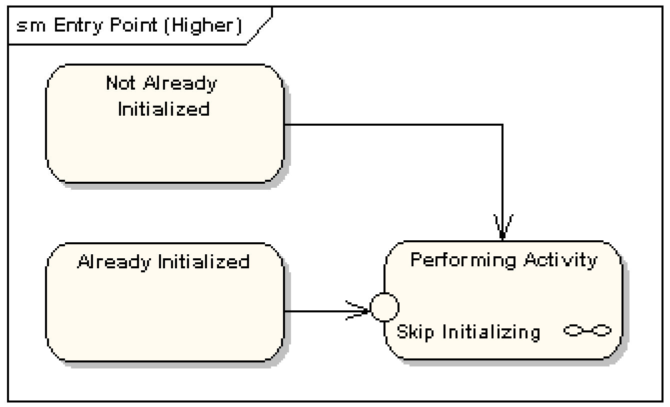
A state machine diagram may include sub-machine diagrams, as in the example below.

# Entry Point

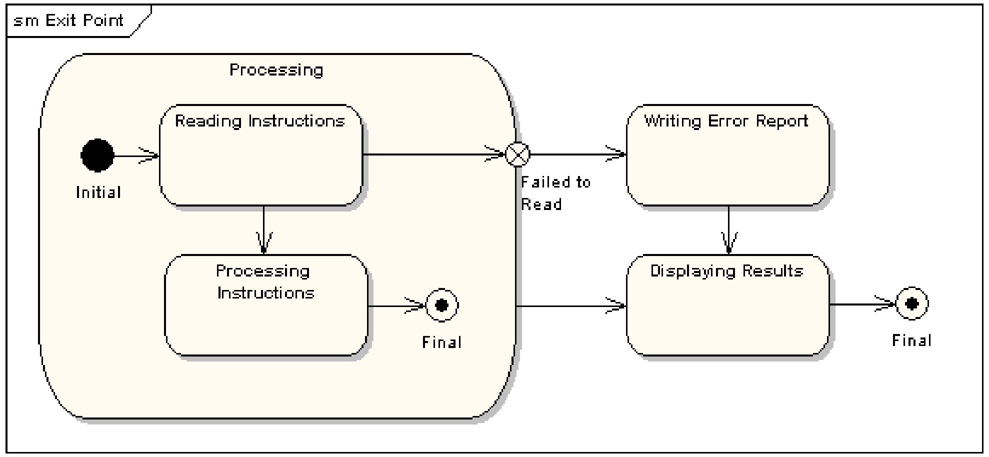


Sometimes you won’t want to enter a sub-machine at the normal initial state. For example, in the following sub-machine it would be normal to begin in the "Initializing" state, but if for some reason it wasn’t necessary to perform the initialization, it would be possible to begin in the "Ready" state by transitioning to the named entry point.

The following diagram shows the state machine one level up.

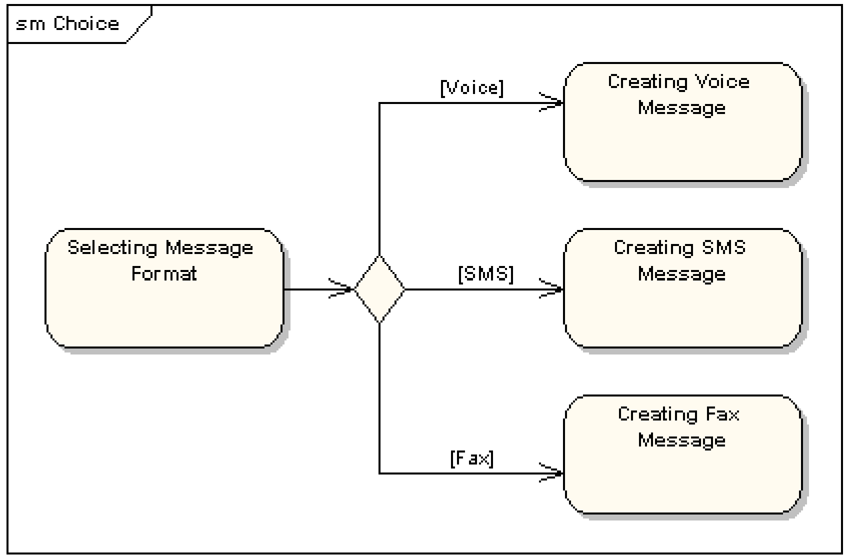


# Exit Point



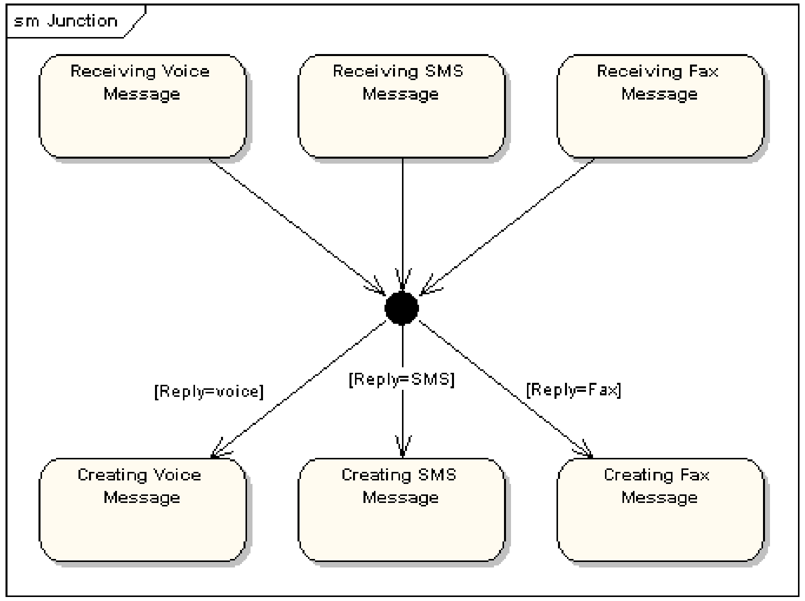
In a similar manner to entry points, it is possible to have named alternative exit points. The following diagram gives an example where the state executed after the main processing state depends on which route is used to transition out of the state.

# Choice Pseudo-State



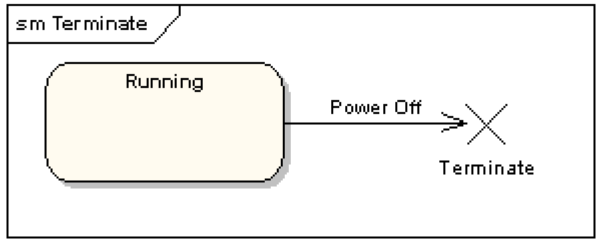
A choice pseudo-state is shown as a diamond with one transition arriving and two or more transitions leaving. The following diagram shows that whichever state is arrived at, after the choice pseudo-state, is dependent on the message format selected during execution of the previous state.

## Junction Pseudo-State



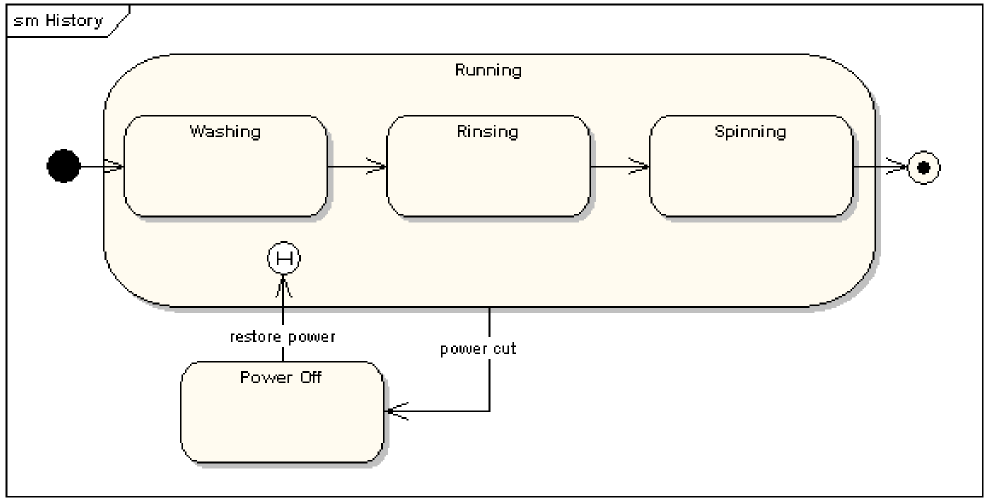
Junction pseudo-states are used to chain together multiple transitions. A single junction can have one or more incoming, and one or more outgoing, transitions; a guard can be applied to each transition. Junctions are semantic-free. A junction which splits an incoming transition into multiple outgoing transitions realizes a static conditional branch, as opposed to a choice pseudo-state which realizes a dynamic conditional branch.

## Terminate Pseudo-State



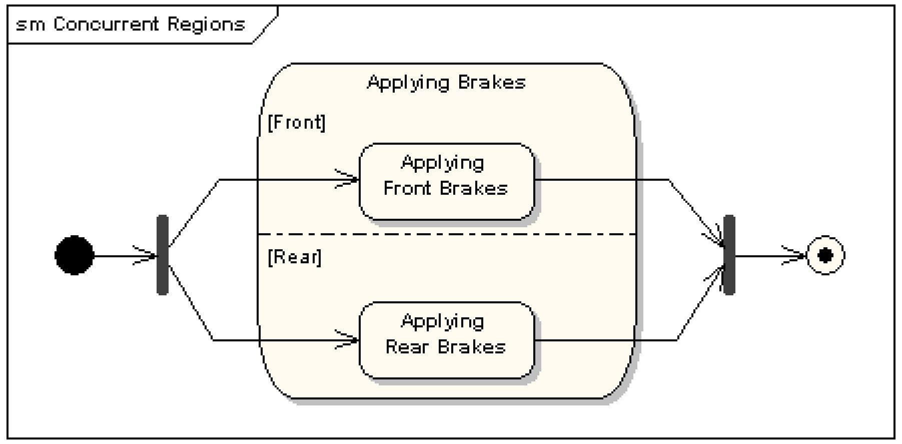
Entering a terminate pseudo-state indicates that the lifeline of the state machine has ended. A terminate pseudo-state is notated as a cross.

# History States



A history state is used to remember the previous state of a state machine when it was interrupted. The following diagram illustrates the use of history states. The example is a state machine belonging to a washing machine.

# Concurrent Regions



A state may be divided into regions containing sub-states that exist and execute concurrently. The example below shows that within the state "Applying Brakes", the front and rear brakes will be operating simultaneously and independently. Notice the use of fork and join pseudo-states, rather than choice and merge pseudo-states. These symbols are used to synchronize the concurrent threads.

Thank you