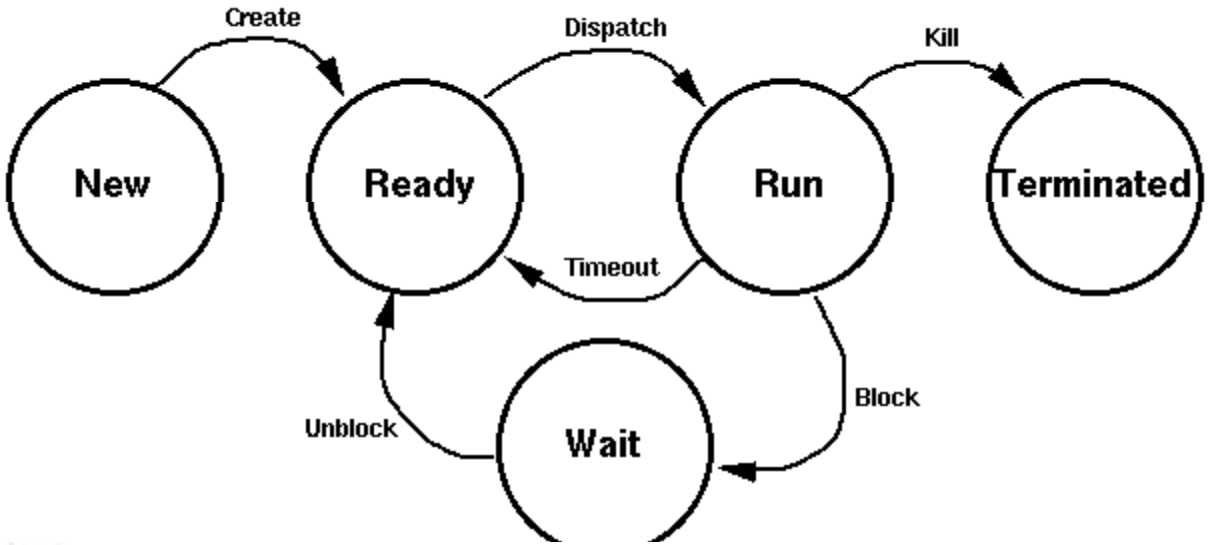
STATE MACHINE DIAGRAM

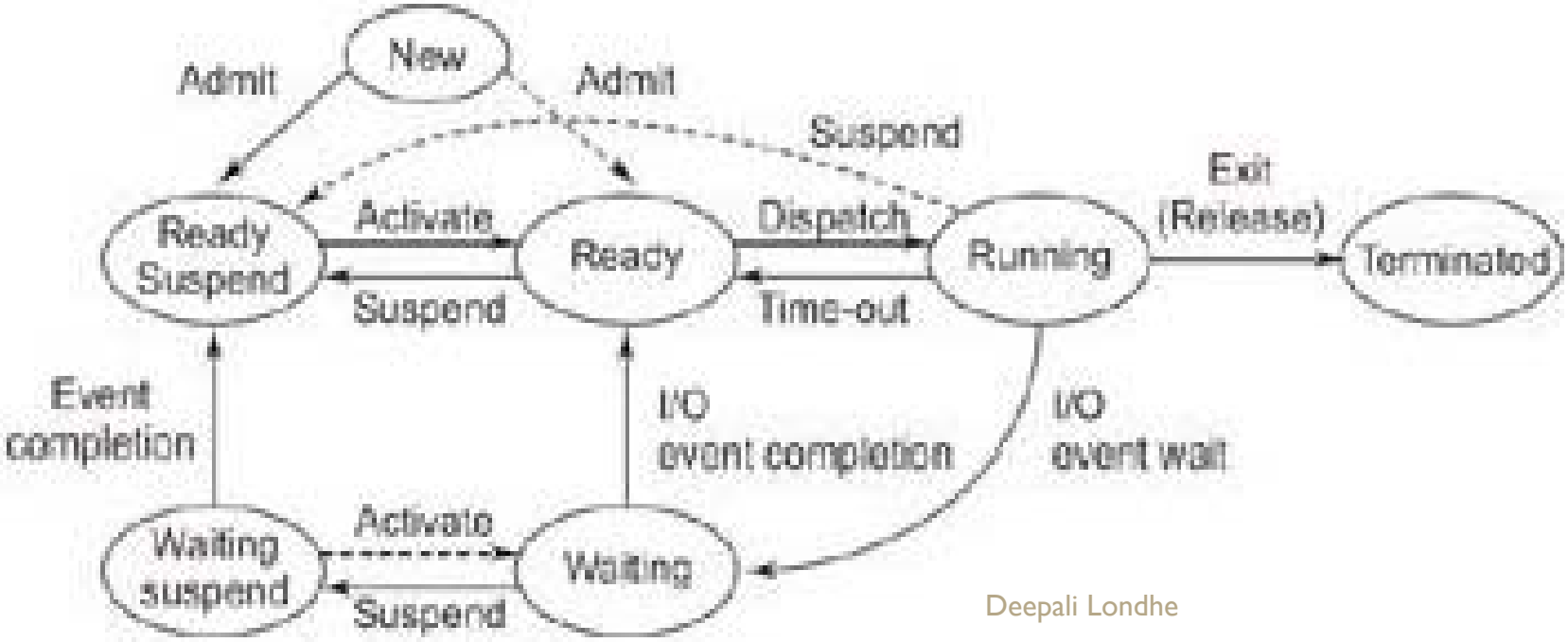
State chart Diagram/ State Diagram

-Deepali Londhe

State Diagrams

OS (Process) States





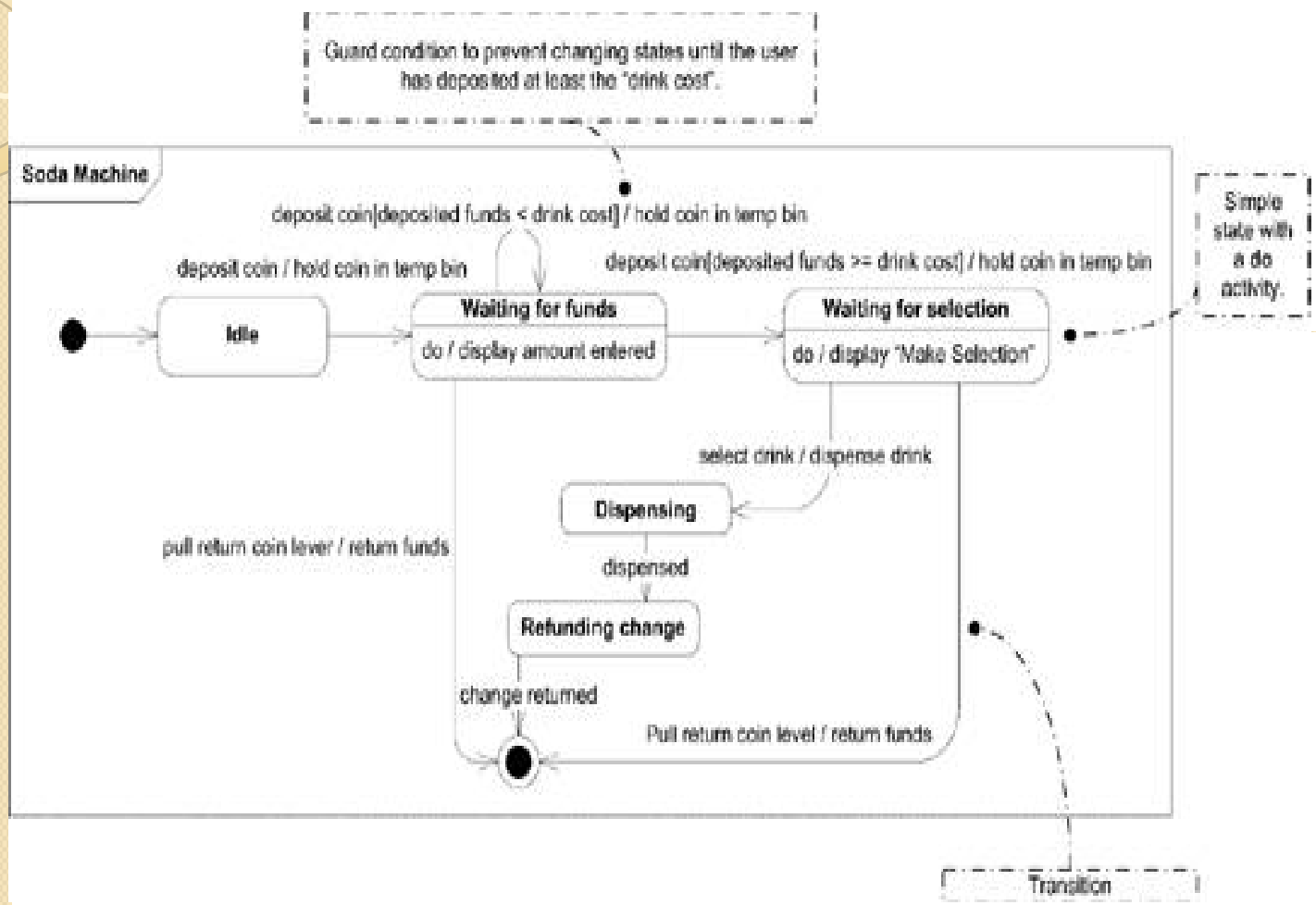
State Machine

- State machine diagrams capture the behavior of a software system.
- Used to model the behavior of a class, subsystem, or entire application.
- UML has two types of state machines:
  - *Behavioral state machines*
  - *Protocol state machines*

State Machine

- *Behavioral state machines*
  - Show the behavior of model elements such as objects.
  - A behavioral state machine represents a specific implementation of an element.
- *Protocol state machines*
  - Show the behavior of a protocol.
  - Protocol state machines aren't typically tied to a particular implementation, and they show the required protocol behavior.
- Behavioral and protocol state machines share common elements; however, PSM are not tied to an implementation and have restrictions on their transitions.
- PSMs are a specialization of behavioral state machines

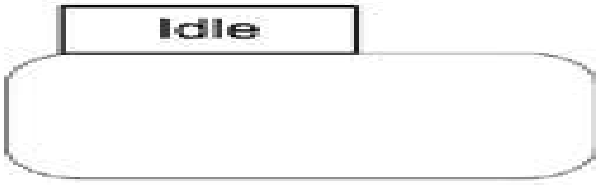
Behavioral state machines



State

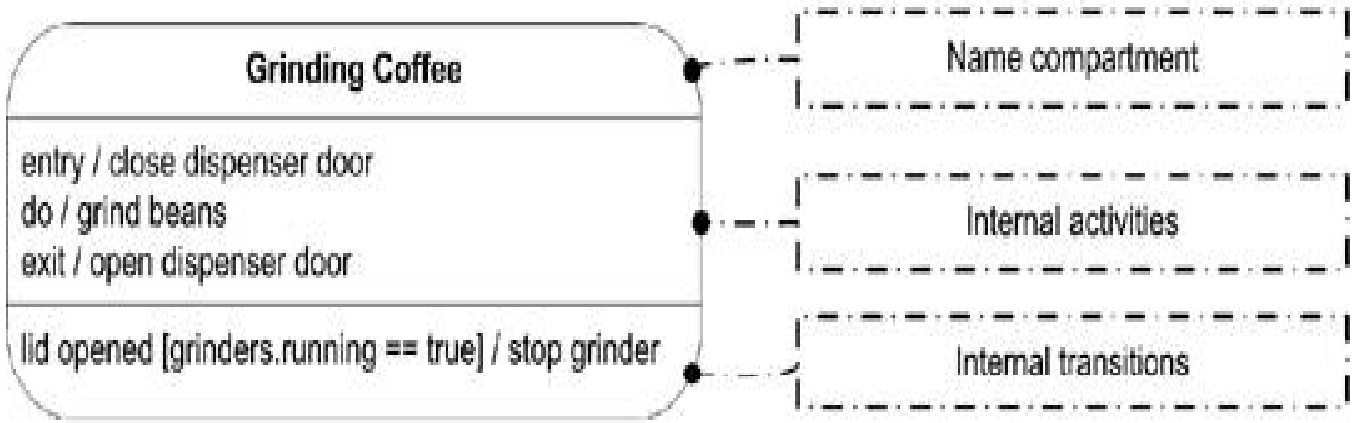
- States model a specific moment in the behavior of a classifier.
- This moment in time is defined by some condition being true in the classifier.
- States model a situation in the behavior of a classifier when an *invariant condition holds true*.
- A state can represent a static situation, such as
- "Waiting for Username" or a dynamic situation where the state is actively processing data, such as "Encrypting Message."

A simple state



A state with its name in a tab

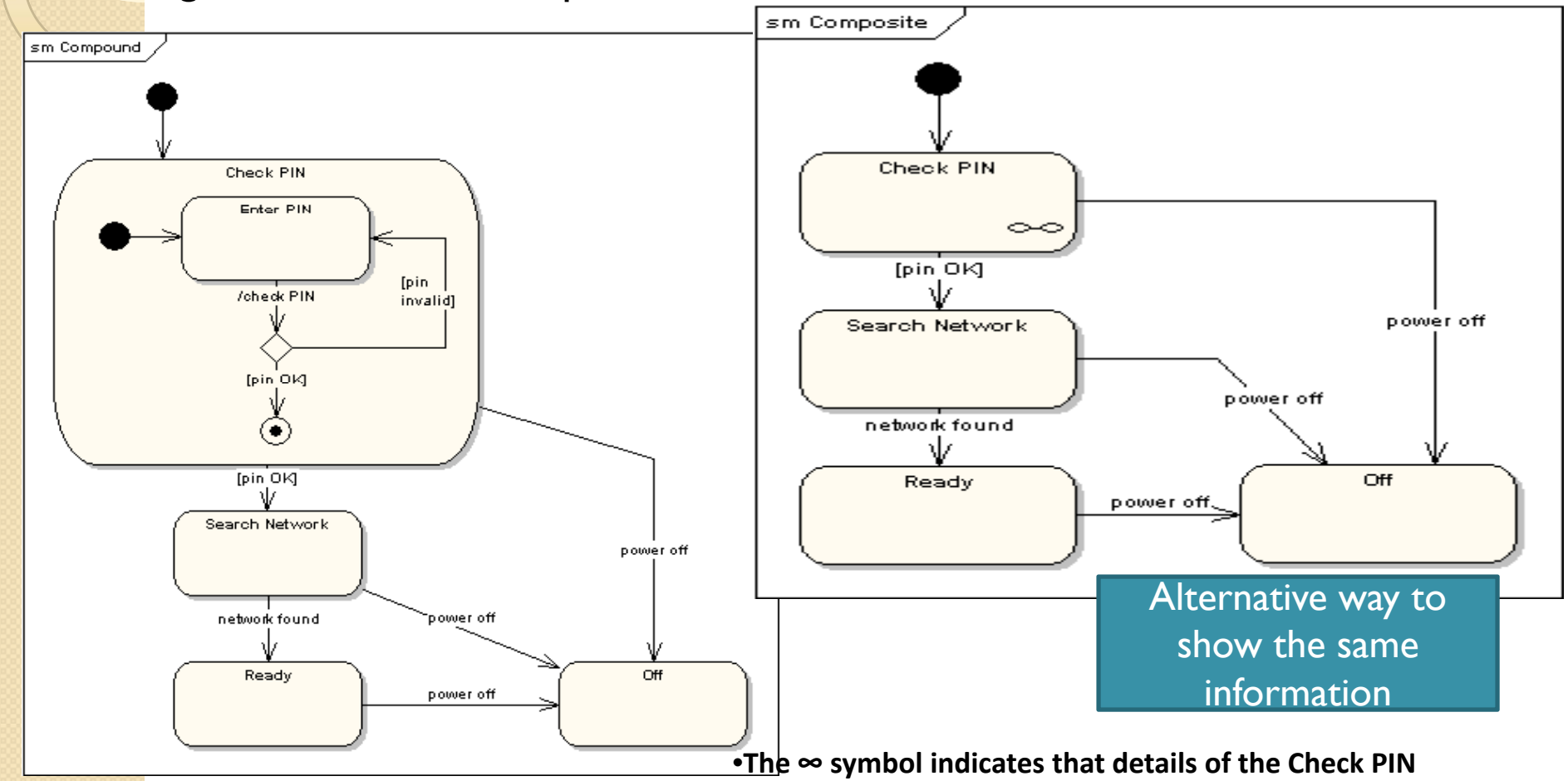
- *Name*
  - Shows the name of the state. This should not be used if the name is placed in a tab.
- *Internal activities*
  - Shows a list of internal activities that are performed while in the state. See "Activities" for the syntax.
- *Internal transitions*
  - Shows a list of internal transitions (see "Transitions") and the events that trigger them.
  - Write an internal transition as:
    - ***event (attributeList) [guard condition]/ transition***



A state with compartments

State Machine Diagrams

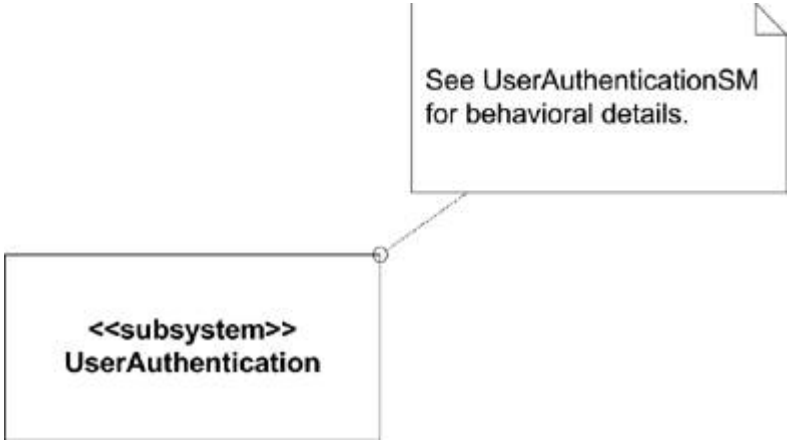
- ***Compound States*** **-** A state machine diagram may include sub-machine diagrams, as in the example below.



•The ∞ symbol indicates that details of the Check PIN sub-machine are shown in a separate diagram.

Example of linking a state machine to a classifier

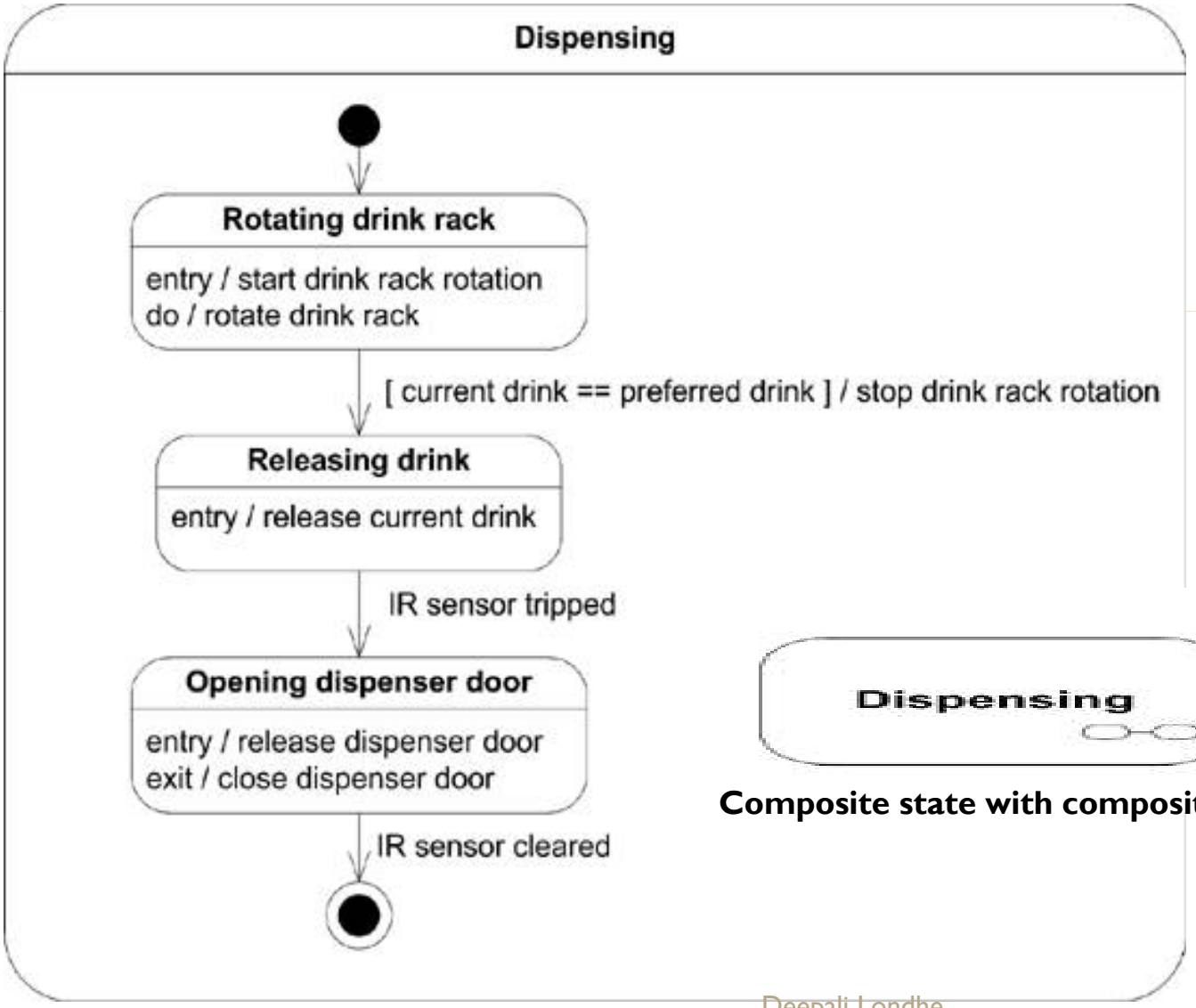
A state machine is often associated with a classifier in the larger UML model. for example, a class or subsystem. However, UML doesn't define a specific notation to show this relationship.
One possible notation is to use a note labeled with the name of the state machine and linked to the classifier.



Composite State

- A composite state is a state with one or more regions.
- A *region is simply a container for substates.*
- A composite state with two or more regions is called *orthogonal.*
- A composite state may have an additional compartment called the decomposition compartment.
- A *decomposition compartment is a detailed view of the composite state where you can show a*composite state's regions, substates, and transitions.

Composite state with one region



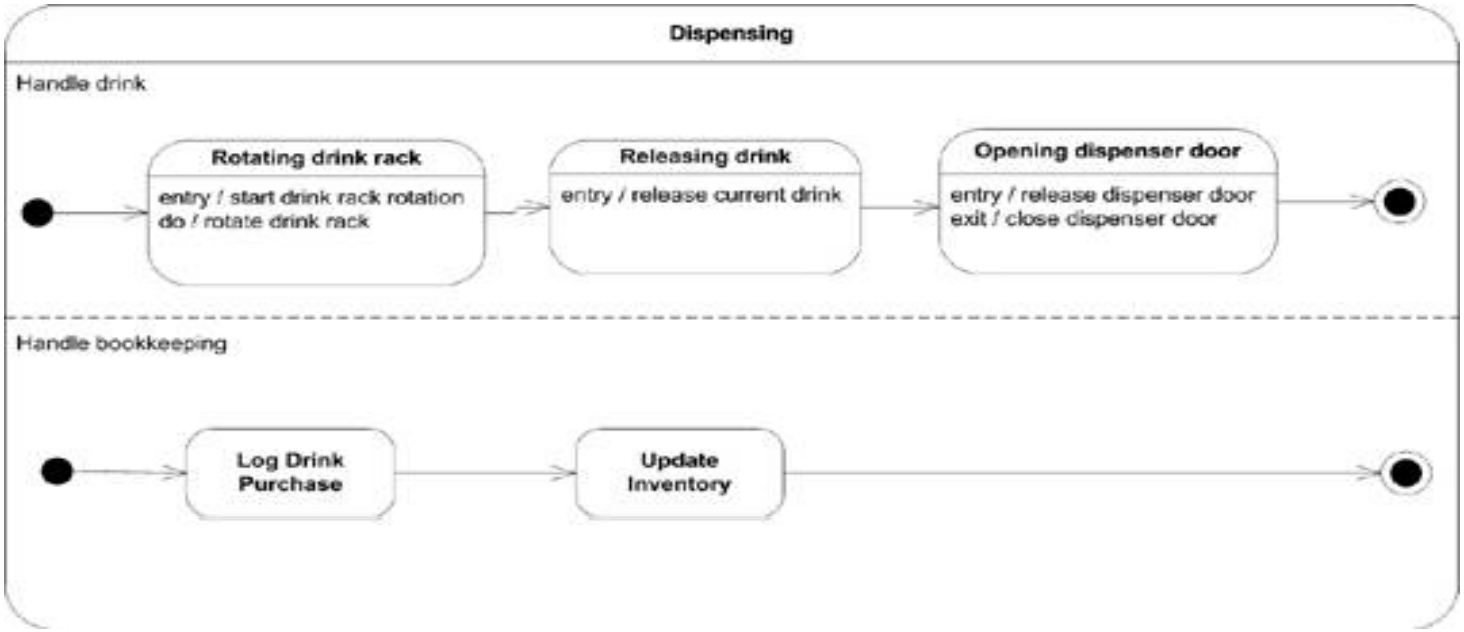
Composite state with composite icon

Regions

•A region is shown using a dashed line dividing the decomposition compartment. You may name each region by writing its name within the region's area.

•Each region has its own *initial pseudostate and a final state. A transition to a* composite state is a transition to the initial pseudostate in each region.

•Each region within a composite state executes in parallel, and it is perfectly acceptable for one region to finish before another.



A composite state with two regions

Submachine States

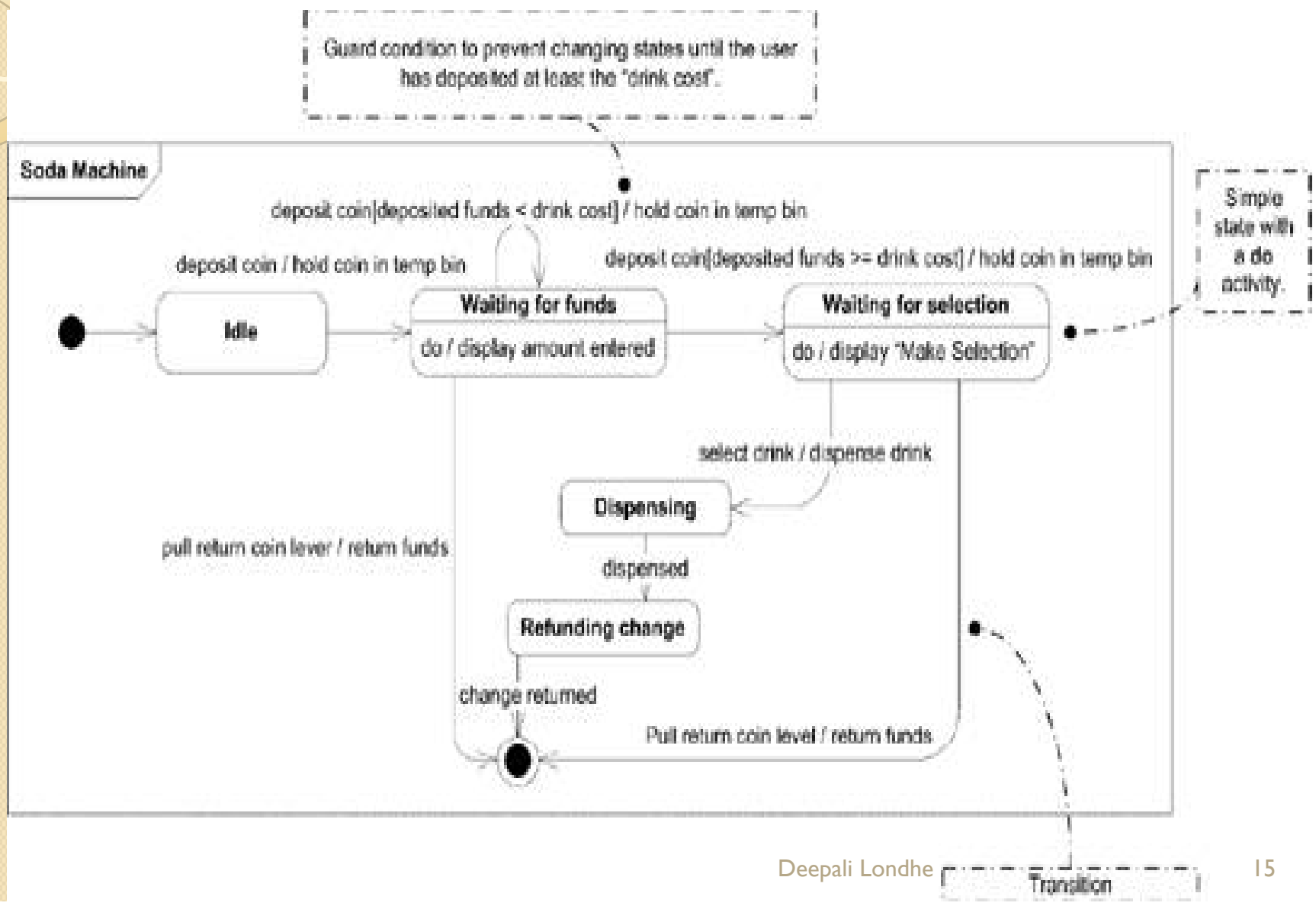
- Submachine states are semantically equivalent to composite states in that they are made up of internal substates and transitions.
- UML defines a submachine state as a way to encapsulate states and transitions so that they can be reused.
- A submachine state simply means that another statemachine, a submachine state machine, is contained by the state.
- A submachine state is shown in the same rounded rectangle as any other state, except you show the name of the state, followed by a colon (:), followed by the name of the referenced submachine.

A submachine state referencing the credit card validation state machine

Transitions

- A transition shows the relationship, or path, between two states or pseudostates. It represents the actual change in the configuration of a state machine as it heads from one state to the next. Each transition can have a guard condition that indicates if the transition can even be considered (*enabled),* a trigger that causes the transition to execute if it is enabled, and any effect the transition may have when it occurs.
- ***Syntax:*** ***trigger [guard] / effect***
- ***Trigger:*** Indicates what condition may cause this transition to occur. The trigger is typically the name of an event, though it may be more complex.
- ***Guard:*** Is a constraint that is evaluated when an event is fired by the state machine to determine if the transition should be enabled. Guards should not have any side effects and must evaluate to a boolean. Guards will always be evaluated before a transition is fired.
- ***Effect:*** Specifies an activity that is executed when a transition happens. This activity can be written using operations, attributes, and links of the owning classifier as well as any parameters of the triggering event.

Behavioral state machines



Activities

- An activity represents some functionality that is executed by a system. A state can have activities that are triggered by transitions to and from the state or by events raised while in the state. A state's activities execute only if the state is active.
- Each activity has a label showing when the activity executes, and an optional activity expression.
- *label / activity expression*
- You can write an activity expression using pseudocode:
  - list.append(keystroke) ; print("*")
- or natural language:
  - record keystroke and show password character

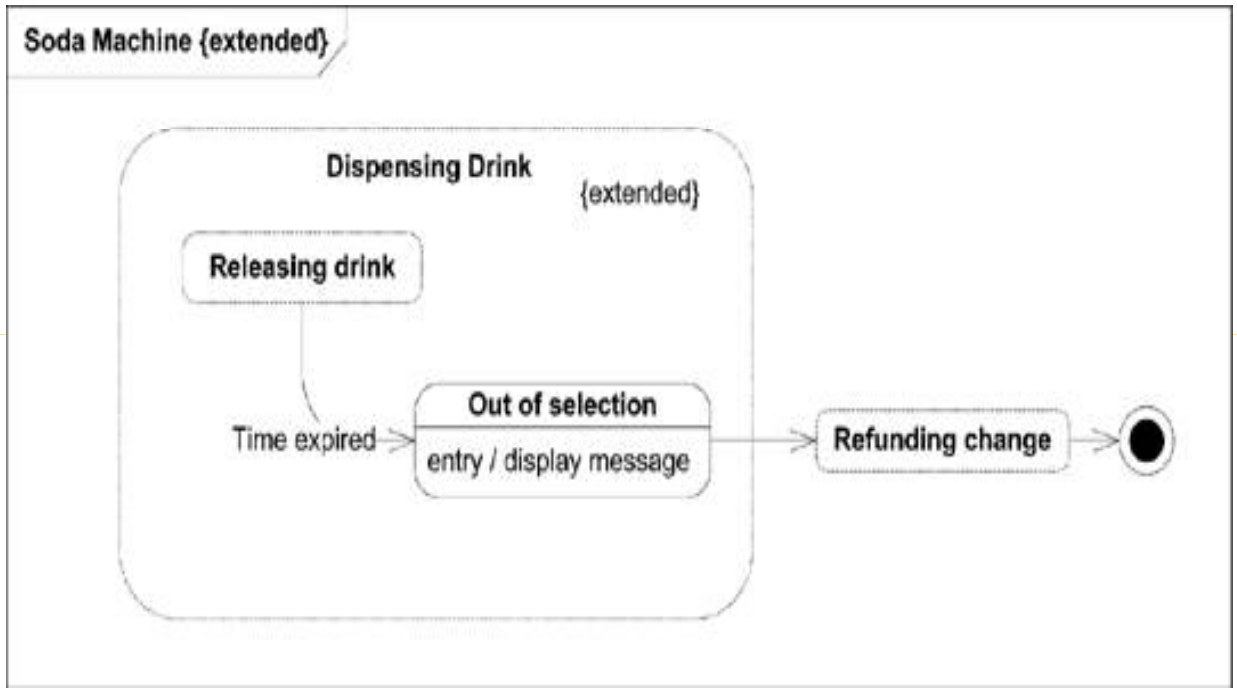
Activities contd..

- UML reserves three activity labels:
- ***Entry***
- Triggers when a state is entered. The entry activity executes before anything else happens in the state.
- ***Exit***
- Triggers when leaving a state. The exit activity executes as the last thing in the state before a transition occurs.
- ***Do***
- Executes as long as a state is active. The do activity executes after the entry activity and can run until it completes, or as long as the state machine is in this state.

State Machine Extension

- Like many other concepts in UML, state machines may be specialized as needed. A specialized state machine is an extension of a general state machine.
- You can specialize a state machine by adding regions, states, pseudostates, or transitions. In addition to adding features to state machines you can redefine states, regions, and transitions.
- When drawing a specialized state machine, draw the inherited states with dashed or gray-toned lines. You may also place the keyword **extended** in curly braces after the name of the state machine.

State Machine Extension

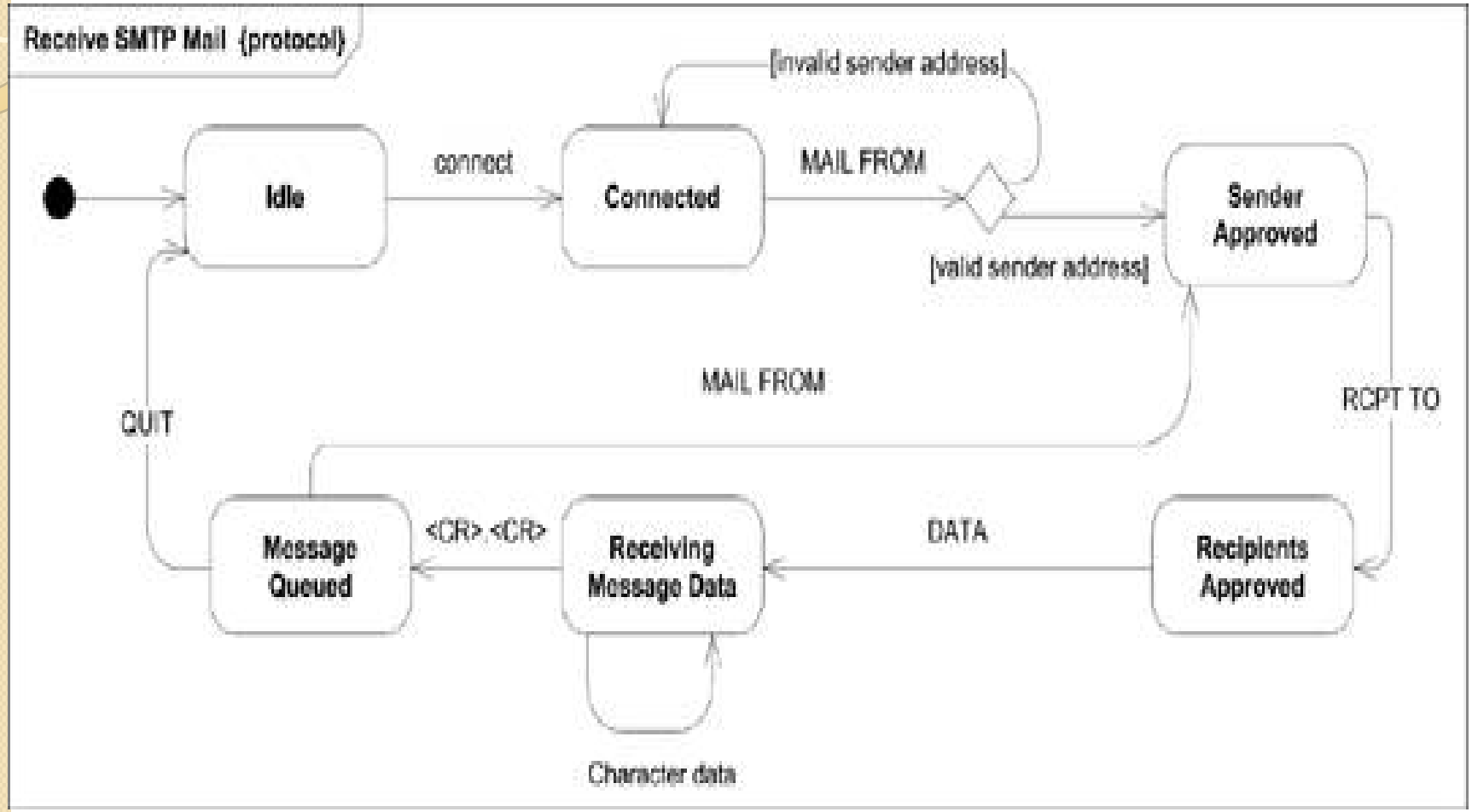


specialized soda dispensing state machine. The `Dispensing Drink` state is extended to introduce a new substate, `Out of selection`. The states `Releasing drink` and `Refunding change` retain their other transitions, and a new transition, `Time expired`, is added to transition to the new substate if the IR sensor isn't triggered.

Protocol State Machines

- Protocol state machines capture the behavior of a protocol, such as HTTP or a challenge-response speakeasy door. They aren't tied to a particular implementation of a protocol; rather, they specify the state changes and events associated with a protocol-based communicati
- Unlike in behavioral state machines, states in protocol state machines represent stable situations where the classifier isn't processing any operation and the user knows its configuration.

A simplified SMTP protocol state machine



Protocol State Machines

- Protocol state machines differ from behavioral state machines in the following ways:
- `entry`, `exit`, and `do` activities can't be used.
- States can have invariants. Place invariants in square brackets under the state name.
- The keyword `protocol` is placed in curly braces after the state machine name to indicate the state machine is a protocol state machine.
- Transitions in protocol state machines have a precondition (in place of the guard in normal transitions), the trigger, and a post condition. The notation for a protocol transition is as follows:
- *`[precondition] event / [postcondition]`*
- Each transition is associated with zero or one operation on the owning classifier. The transition guarantees that the precondition will be `true` before the operation is invoked, and that the post condition will be `true` before entering the target state.
- The effect activity is never specified for a protocol transition.

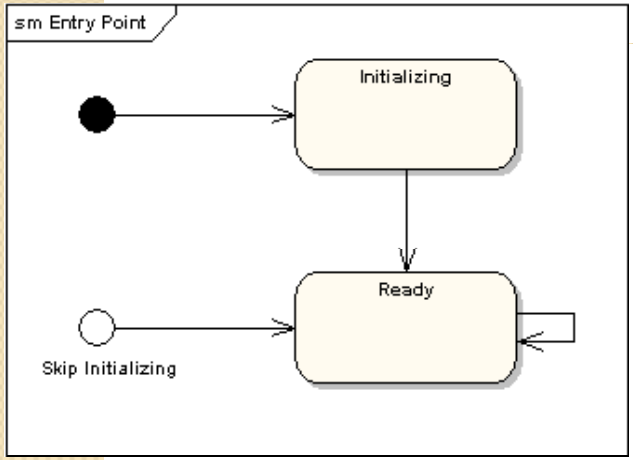
Pseudostates

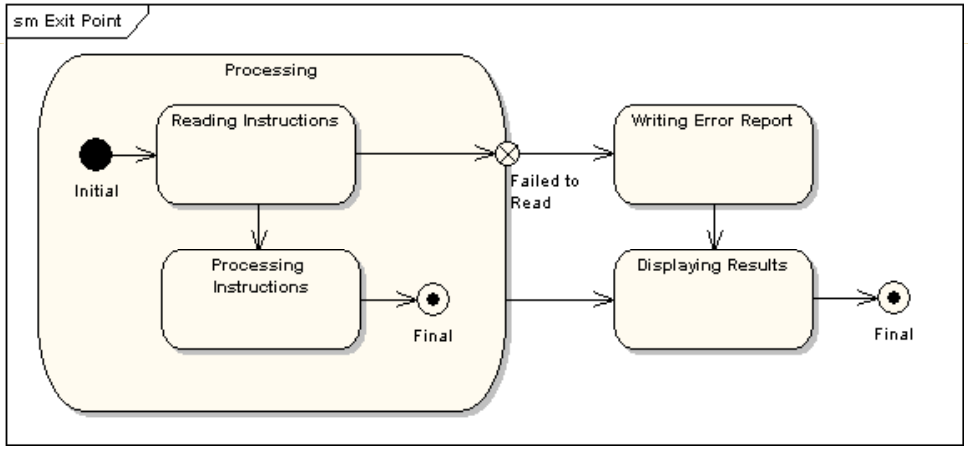
- Pseudostates are special types of states that represent specific behavior during transitions between regular states.

  Combined with basic transitions pseudostates can represent complex state changes within a state machine.
- Ref: sc6

State Machine Diagrams

- ***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.

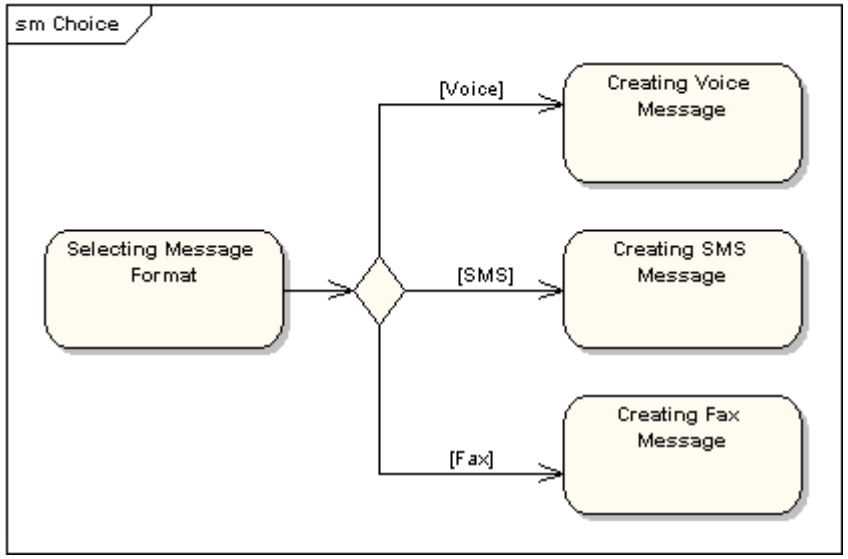




- ***Exit Point*** - In a similar manner to entry points, it is possible to have named alternative exit points. The following diagram gives an example where the state executed after the main processing state depends on which route is used to transition out of the state.

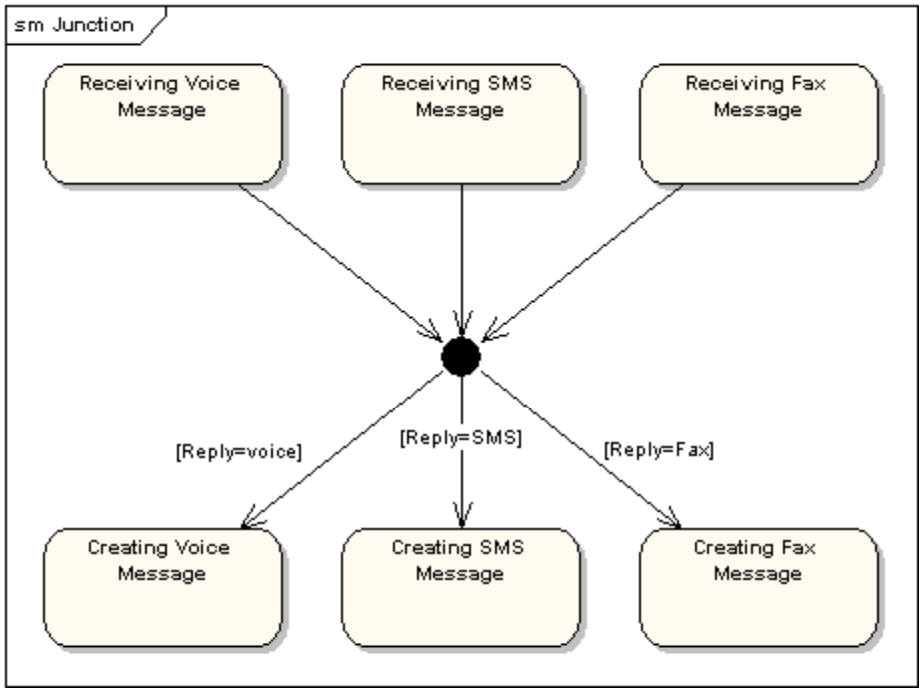
State Machine Diagrams

- ***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.



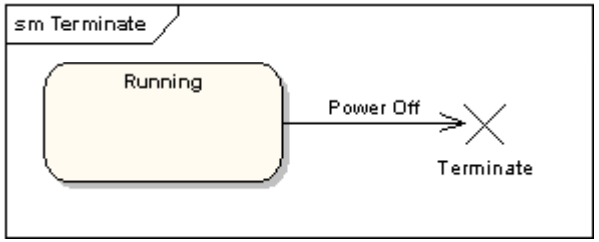
State Machine Diagrams

- ***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.

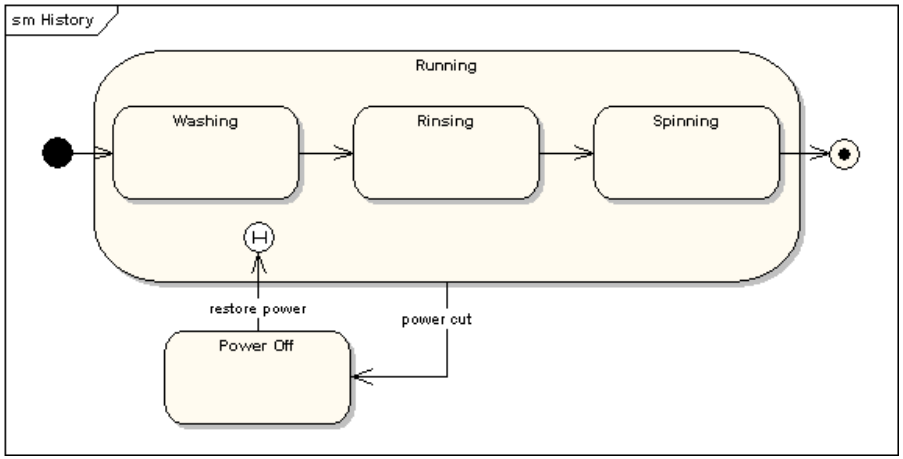


State Machine Diagrams

- ***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.

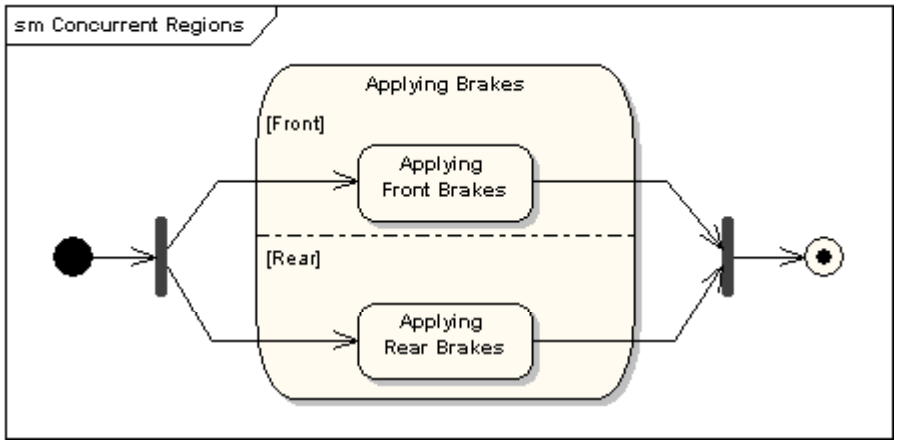


- ***History States*** - A history state is used to remember the previous state of a state machine when it was interrupted. The following diagram illustrates the use of history states. The example is a state machine belonging to a washing machine.

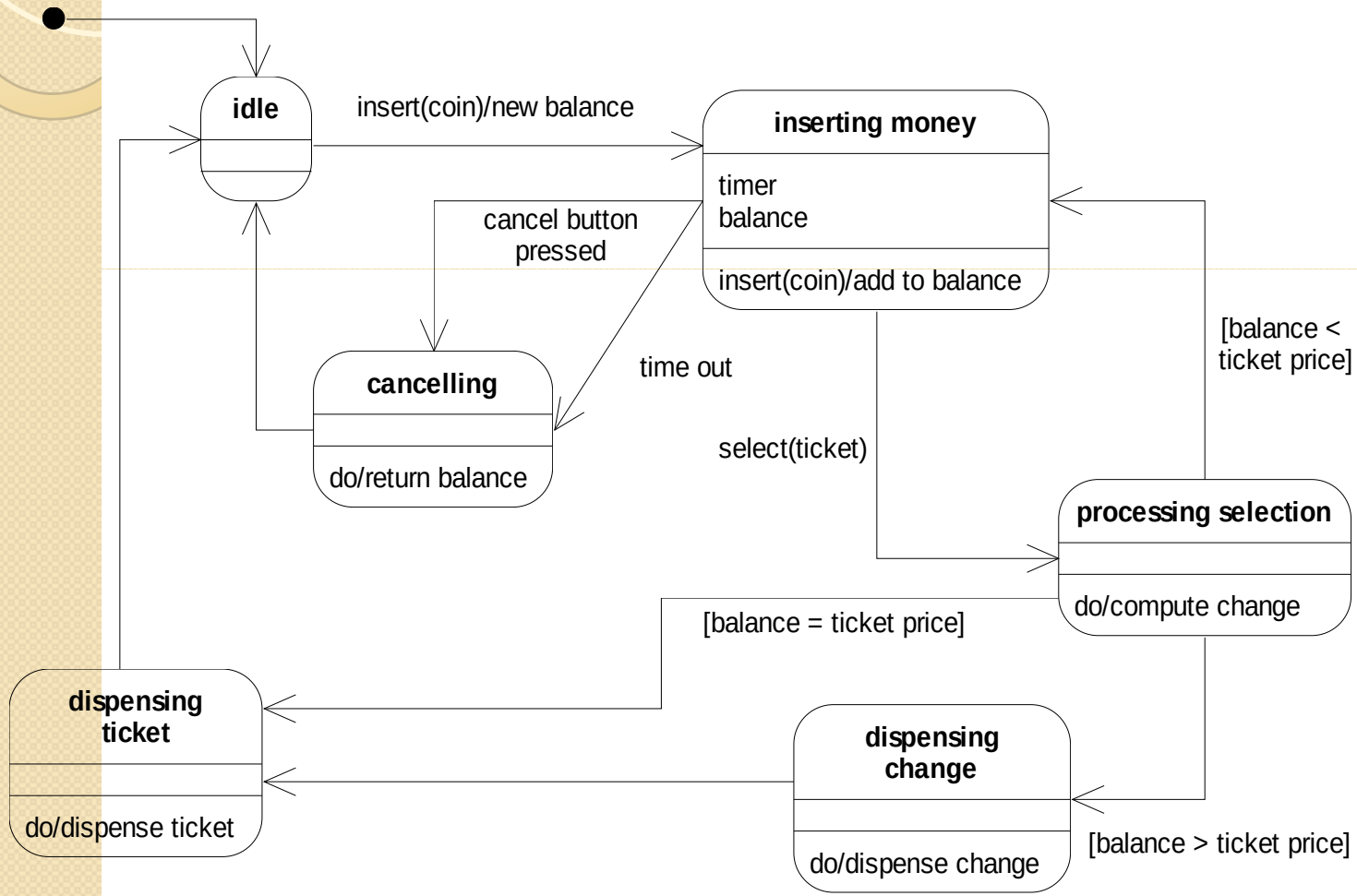


State Machine Diagrams

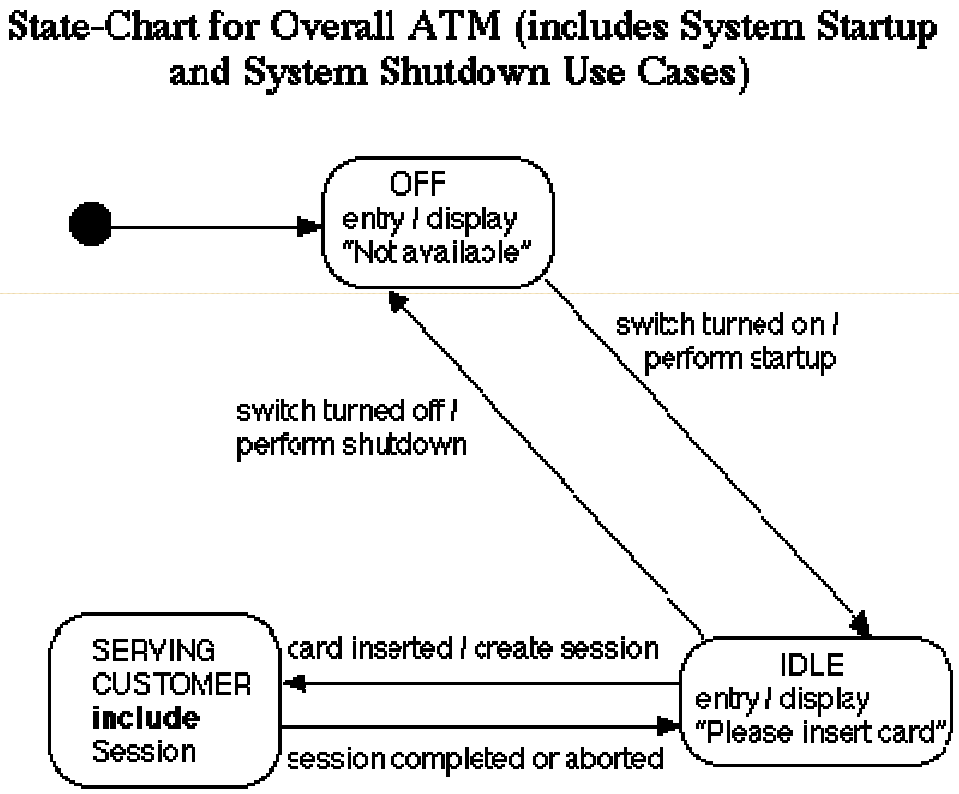
- ***Concurrent Regions*** - A state may be divided into regions containing sub-states that exist and execute concurrently. The example below shows that within the state "Applying Brakes", the front and rear brakes will be operating simultaneously and independently. Notice the use of fork and join pseudo-states,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



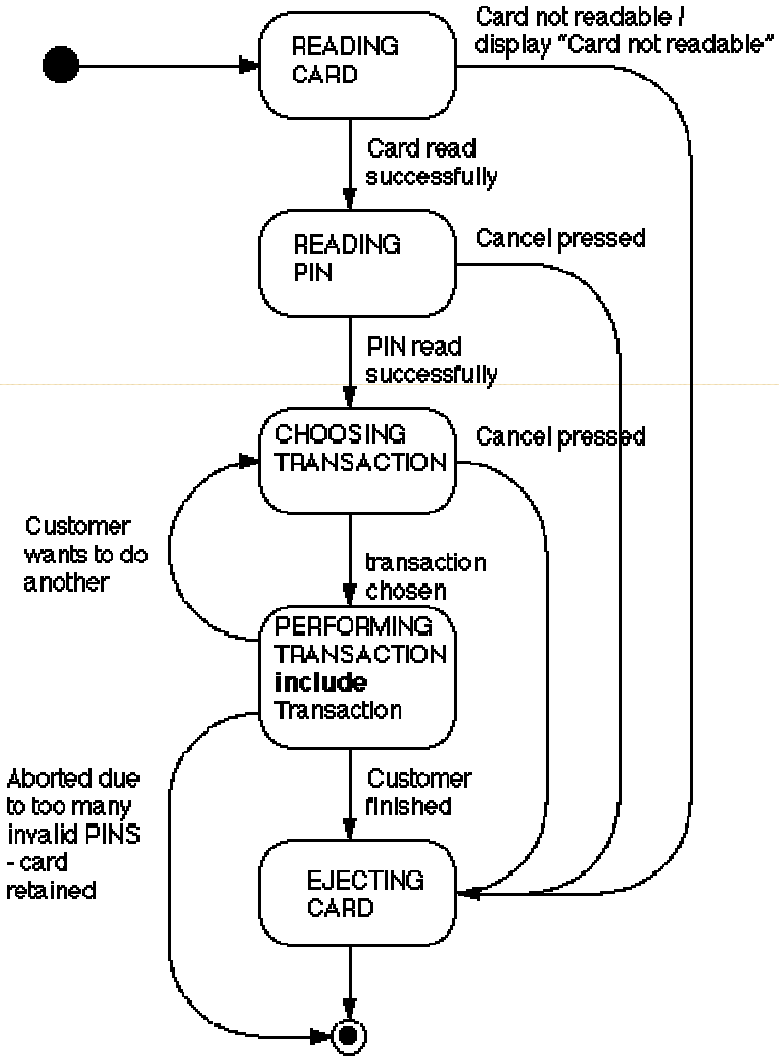
State diagram of ticket machine



ATM SYSTEM SCD



State-Chart for One Session



State-Chart for One Transaction
(italicized operations are unique to each particular type of transaction)