Design - state machines

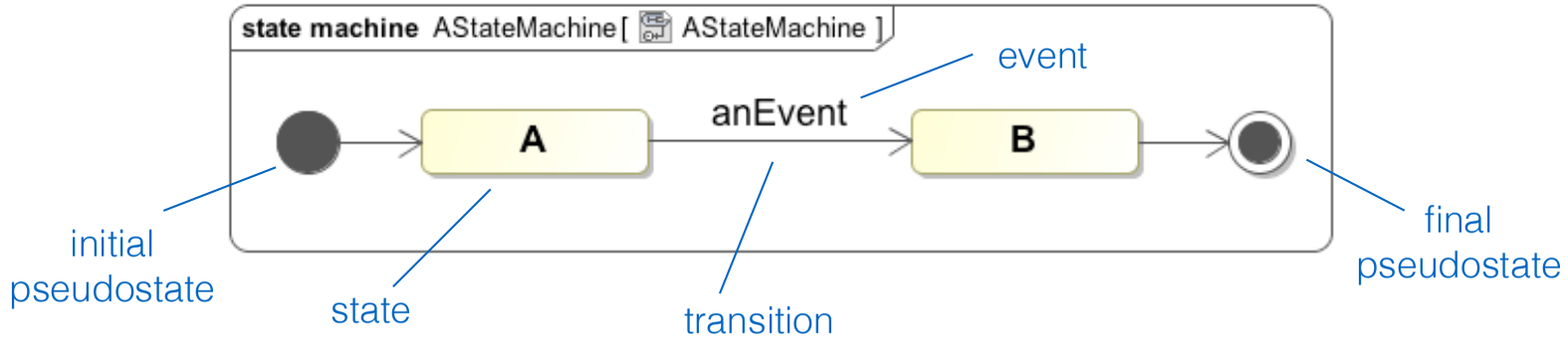
State machines

- Some model elements such as classes, use cases and subsystems, can have interesting dynamic behavior - state machines can be used to model this behavior
- Every state machine exists in the context of a particular model element that:
  - Responds to events dispatched from outside of the element
  - Has a clear life history modeled as a progression of states, transitions and events. We'll see what these mean in a minute!
  - Has current behavior that depends on its past
- A state machine diagram always contains exactly one state machine for one model element

Types of state machine

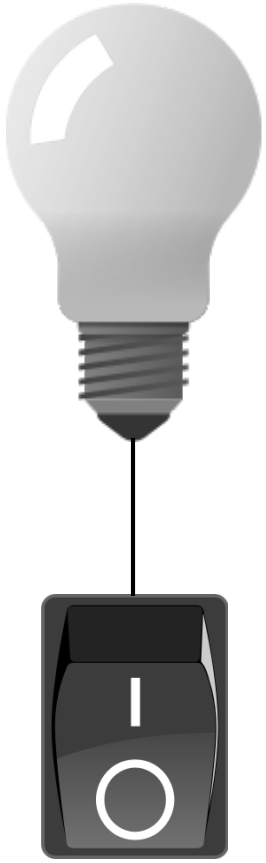
- There are two types of state machines, behavioral and protocol
- Behavior state machines define the behavior of a model element, e.g. the behavior of a classifier, a standalone behavior or an operation
- Protocol state machines model the *protocol* of a classifier:
  - The conditions under which operations of the classifier can be called and the ordering and results of operation calls
- Protocol state machines can model the protocol of classifiers that have no behavior (e.g. interfaces and ports)

Basic state machine syntax

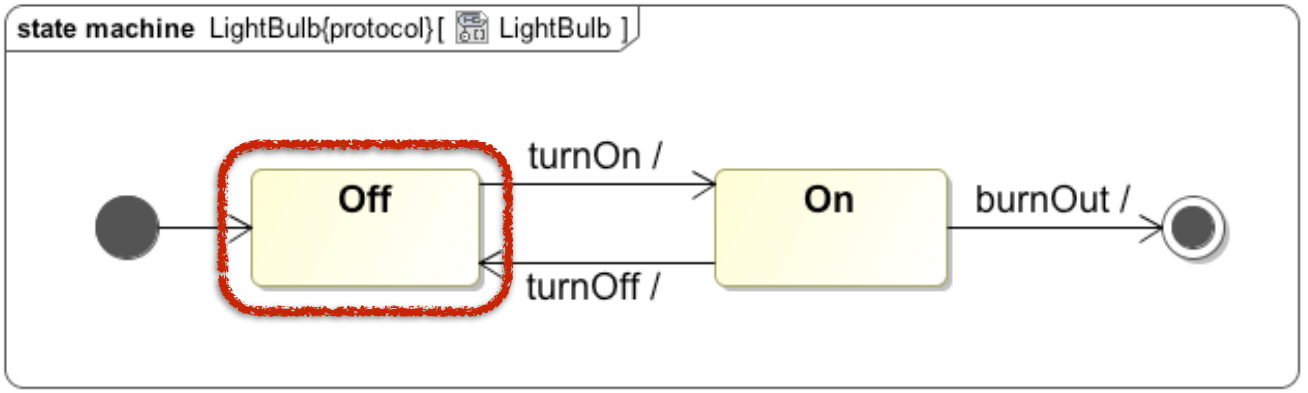


- Every state machine should have a initial pseudostate which indicates the first state of the sequence
- Unless the states cycle endlessly, state machines should have a final state which terminates the sequence of transitions
- We'll look at each element of the state machine in detail in the next few slides!

Light bulb off

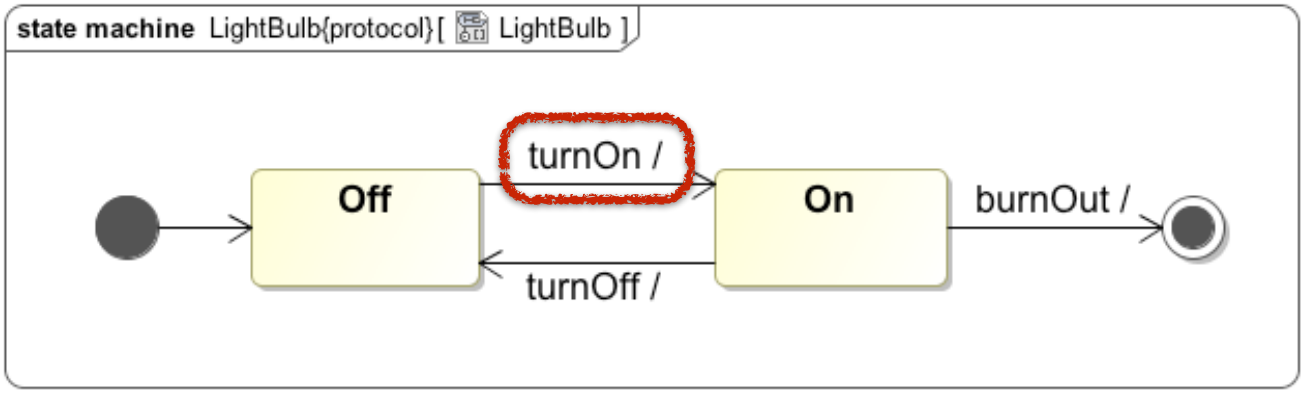


Off



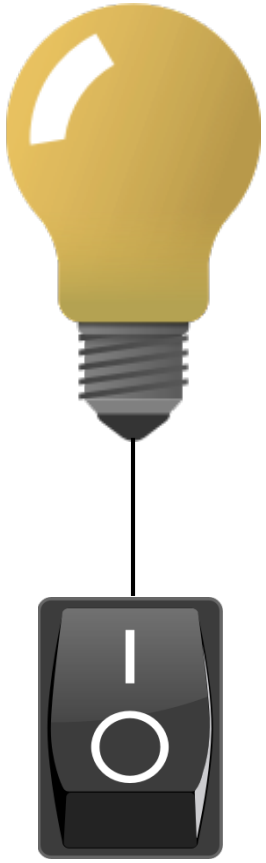
- We begin with the light bulb in the state Off

Turn on light bulb

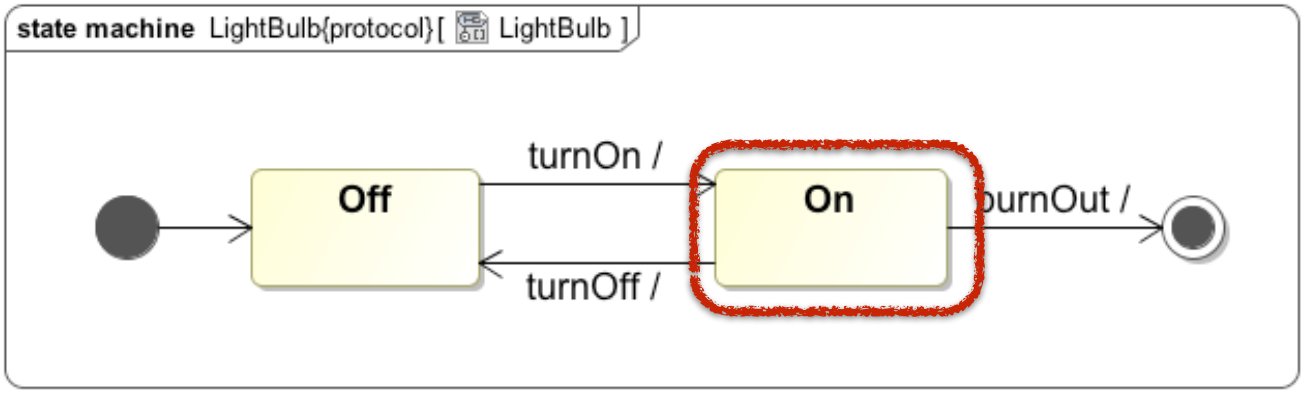


- We throw the switch to On and the event turnOn is sent to the lightbulb

Light bulb on

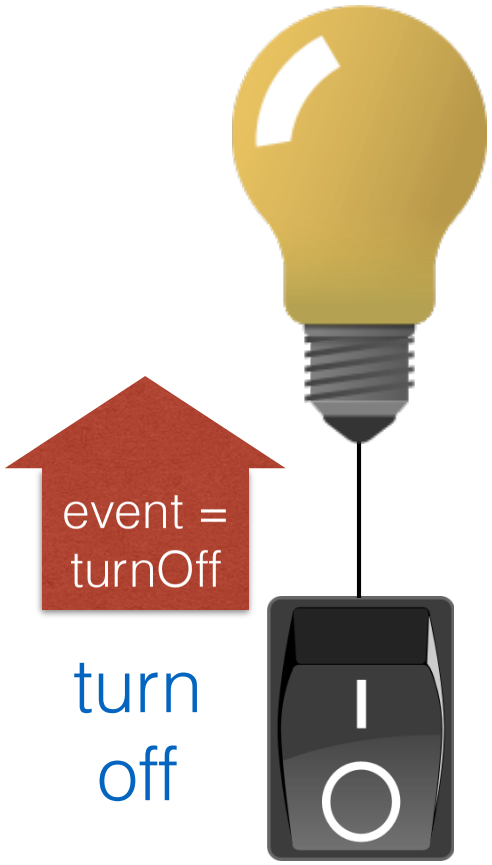


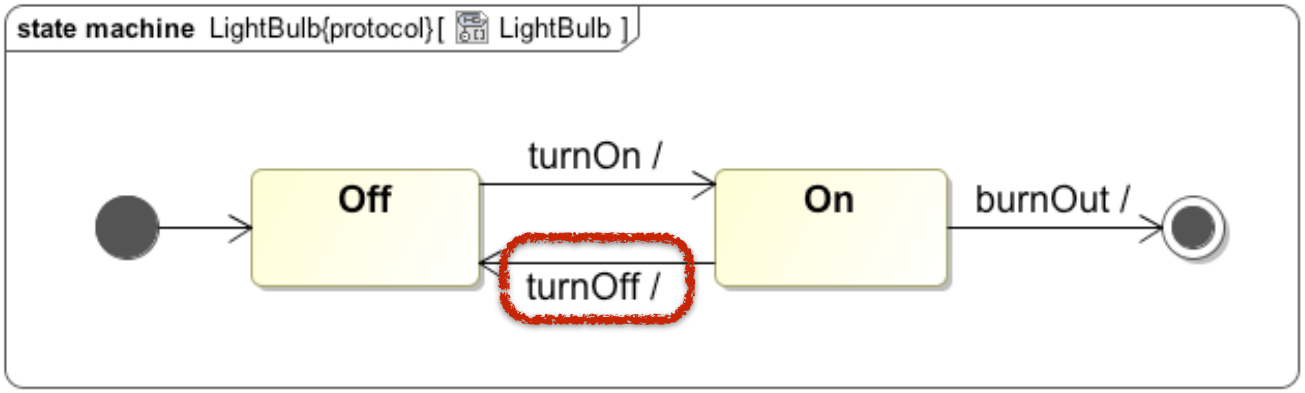
On



- The light bulb is on

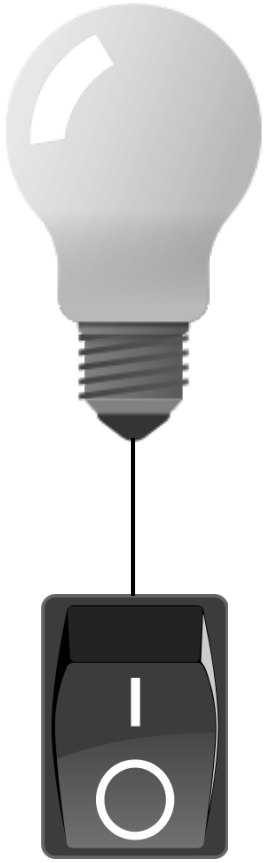
Turn off light bulb



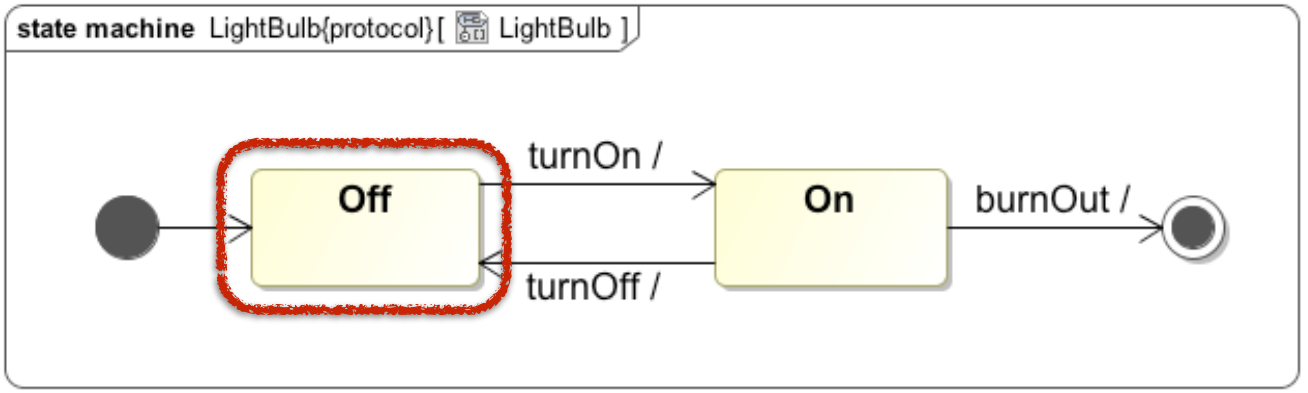


- We turn the switch to Off. The event turnOff is sent to the light bulb

Light bulb off

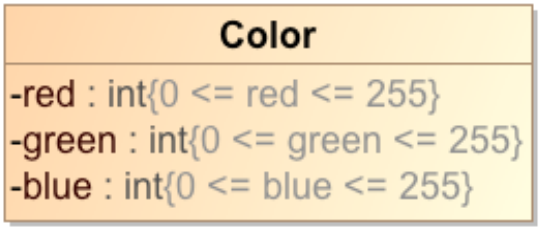


Off



- The light bulb is off

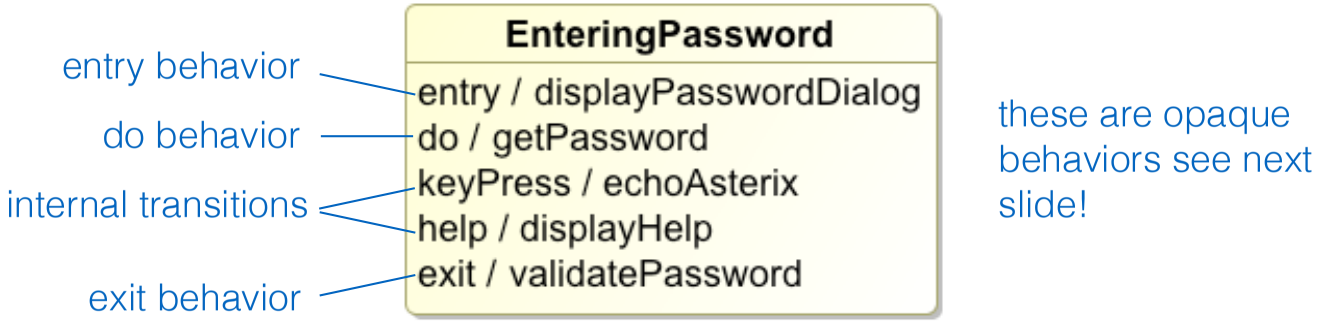
What is a state?



How many states can objects of type Color have?

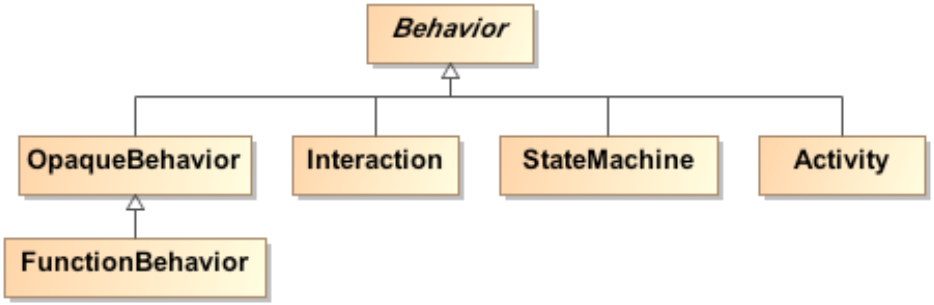
- "A condition or situation during the life of an object during which it satisfies some condition, performs some activity or waits for some event"
- The state of an object at any point in time is determined by:
  - The values of its attributes
  - The relationships it has to other objects
  - The activities it is performing
- States should always model *differences that make a difference*

State syntax



- entry behaviors occur on entry to the state and run to completion
- exit behaviors occur on leaving the state and run to completion
- do behaviors take a finite amount of time and are interruptible by events that exit the state
- Internal transitions are triggered by events (keyPress and help) and occur *within* the state. They do *not* transition to a new state, so do not trigger exit behaviors

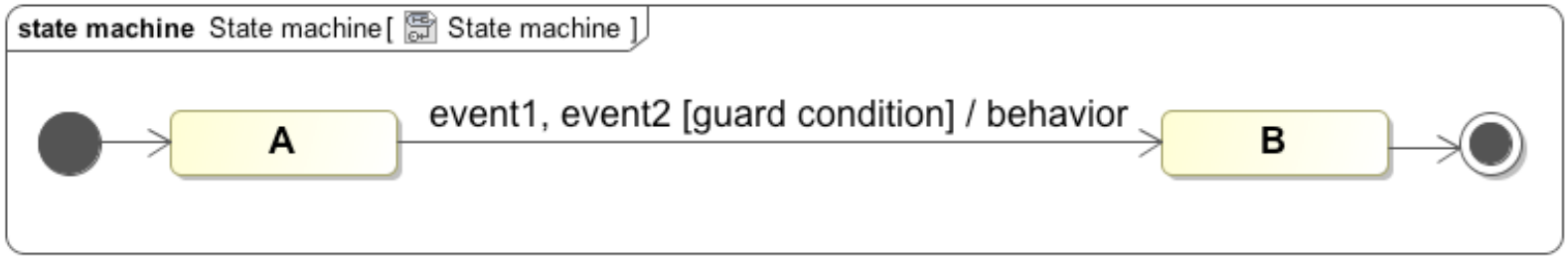
Behaviors



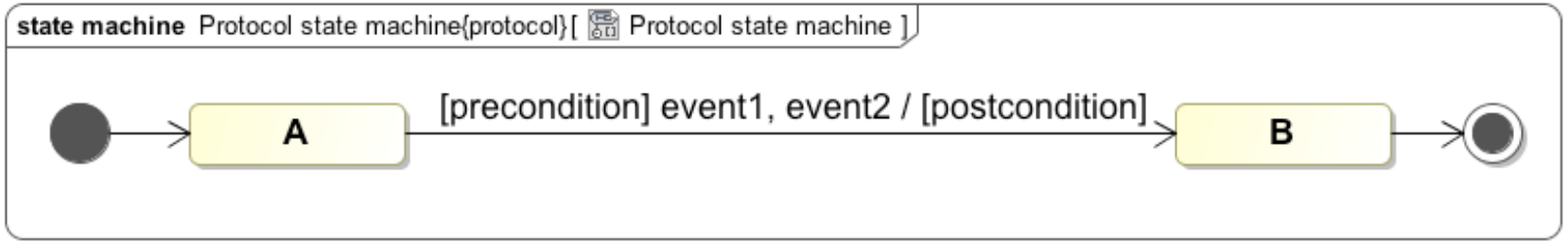
- A behavior describes a set of possible *executions*, where an execution is the performance of one or more *actions*
- Actions can respond to and generate events and access and mutate the state of objects. They can also invoke behaviors. They run to completion. They are atomic *within their context*
- An OpaqueBehavior is specified in a non-UML language such as Java. One behavior may have many specifications in different languages. Behaviors may be mapped to an operation of a classifier
- A FunctionBehavior is an OpaqueBehavior that is a true function, i.e. the set of input parameters are transformed to the set of output parameters *with no other inputs* to the function, *no other outputs* and *no side-effects*

Transition syntax

Behavior state machine (most common)



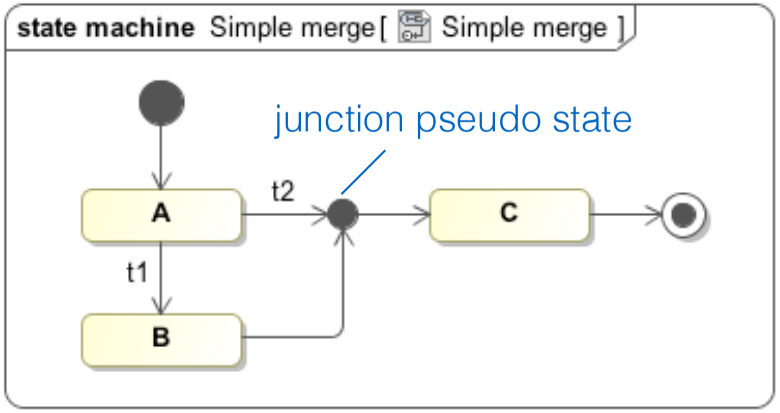
Protocol state machine

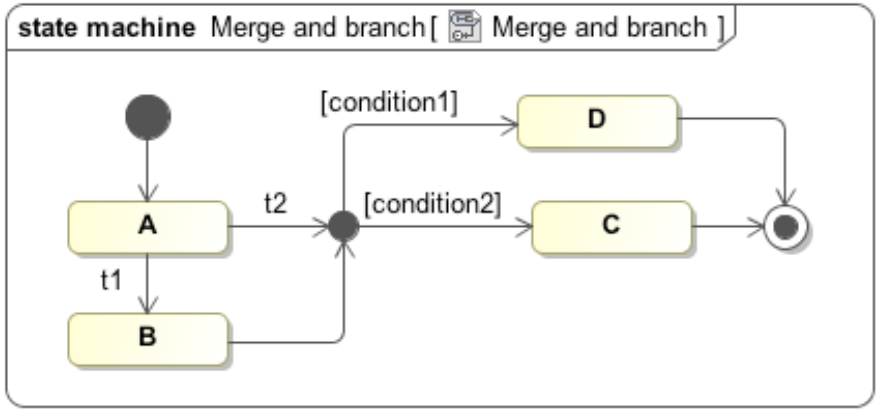


- Events trigger transitions in both types of state machine
- Behavior state machine: The guard condition must be true for the transition to occur. The optional behavior specifies the effect of the transition
- Protocol state machine: The precondition is functionally equivalent to the guard condition. The postcondition must be true after the transition so it is true when state B is entered. If the event is an operation, then it is equivalent to operation pre and post conditions

Junction pseudo state

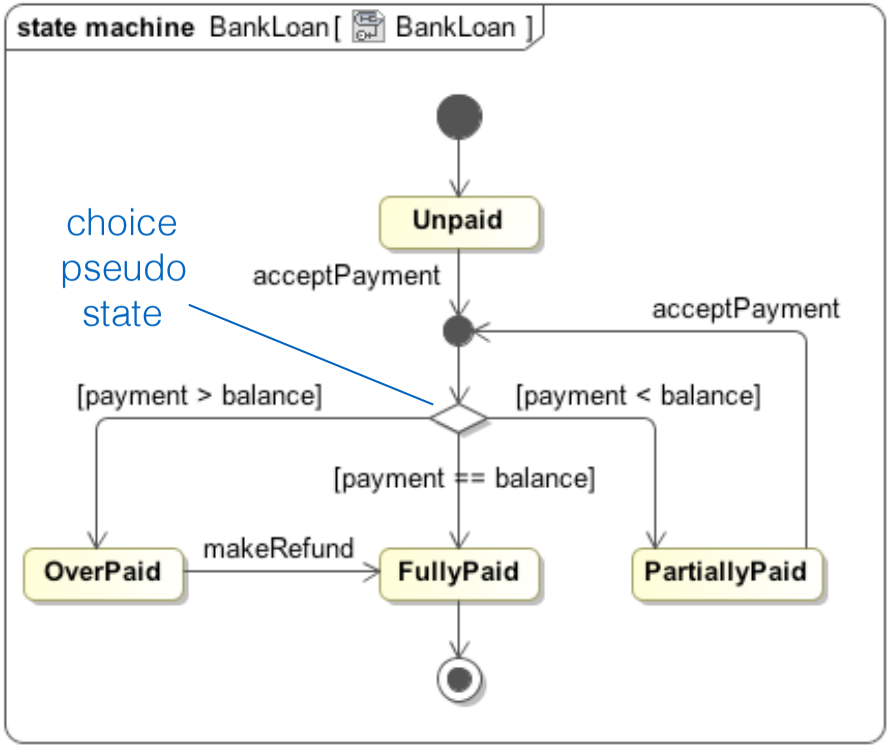
- The junction pseudo state merges transitions together
- If each outgoing transition has a mutually exclusive guard condition, then they can also branch
- This is a *static branch,* i.e. the guard conditions are evaluated *before* any of the merged transitions are traversed





Choice pseudostate

- The choice pseudostate directs incoming transitions to *one* of its outgoing transitions
- This is a *dynamic branch* - the outgoing guard conditions are evaluated at the point when the pseudo state is entered
- Each outgoing transition must have a mutually exclusive guard condition, otherwise it is non-deterministic

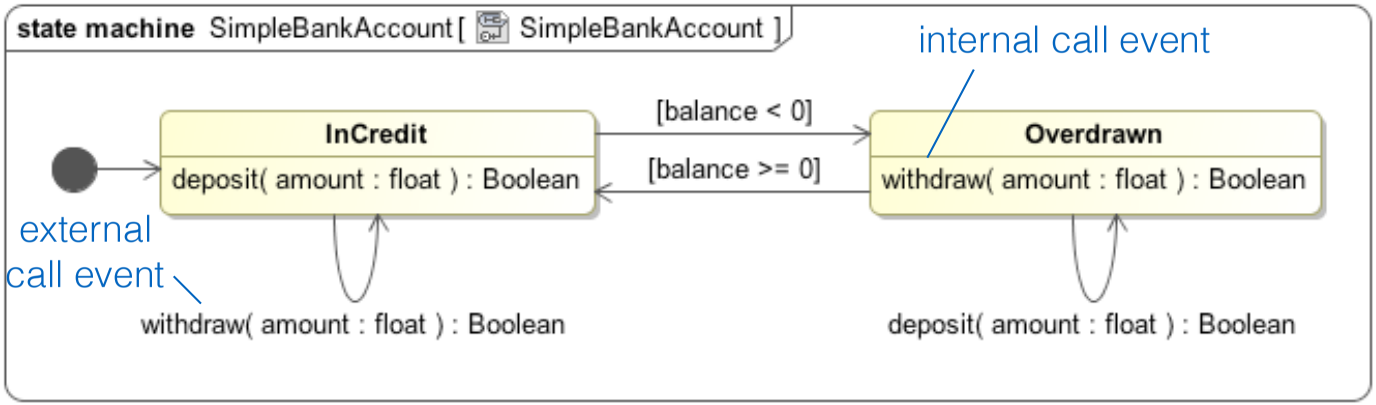


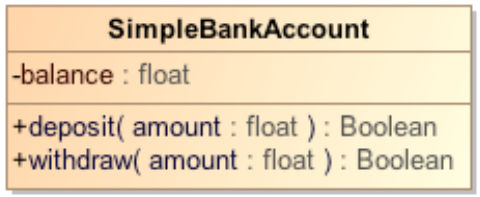
Note: The choice pseudo state can accept one or more input transitions but this can be messy. Instead, we have used a merge pseudostate for neatness

What is an event?

- "The specification of a noteworthy occurrence that has location in time and space". Events trigger transitions in state machines
- Events can be shown externally on transitions, or internally within states (internal transitions)
- There are four types of event: Call event, signal event, change event and time event

Call event

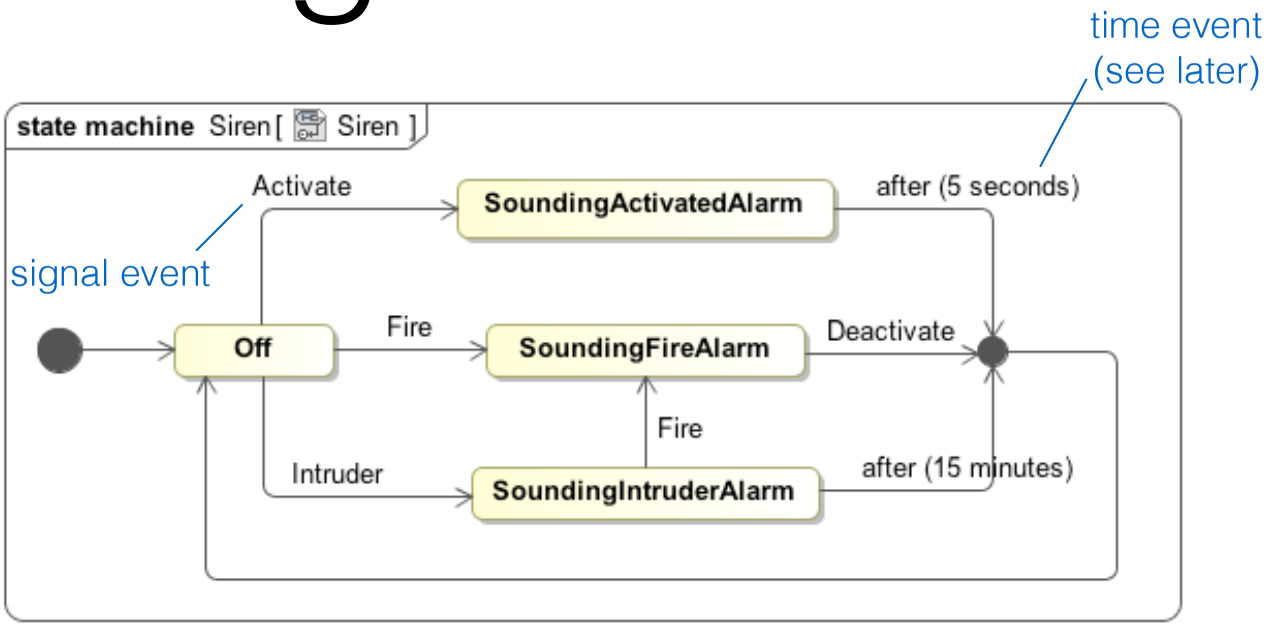




- This is a call for an operation execution on the context classifier. The event must have the same signature as an operation

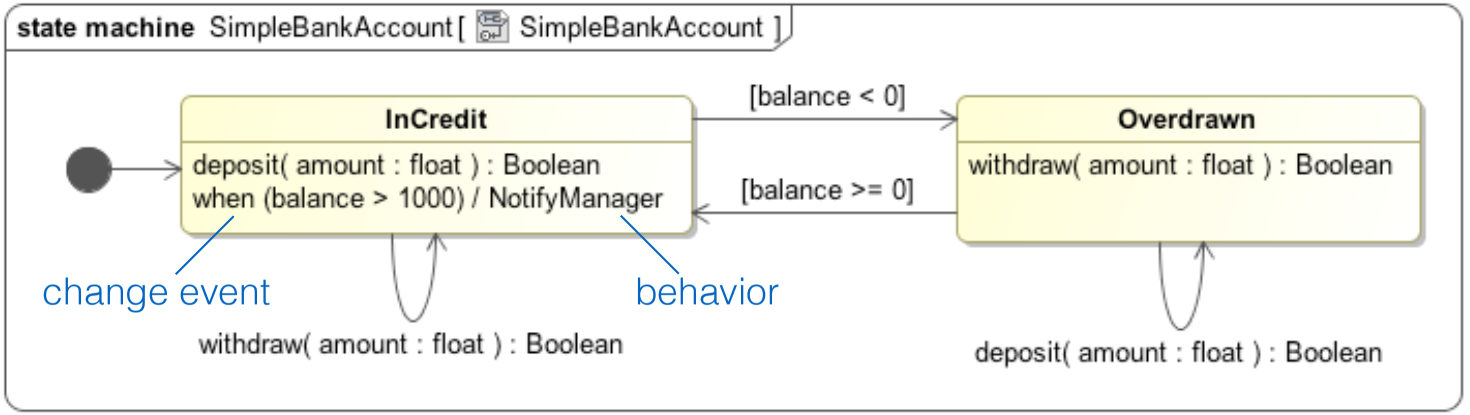
- A sequence of actions may be specified for a call event. The actions can use attributes and operations of the context class and their return value must match the return type of the operation

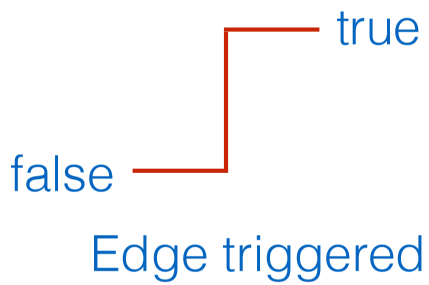
Signal event



- Represents the asynchronous reception of a signal by the context classifier
- Obviously, the context classifier must be capable of receiving signals asynchronously, e.g. an active class

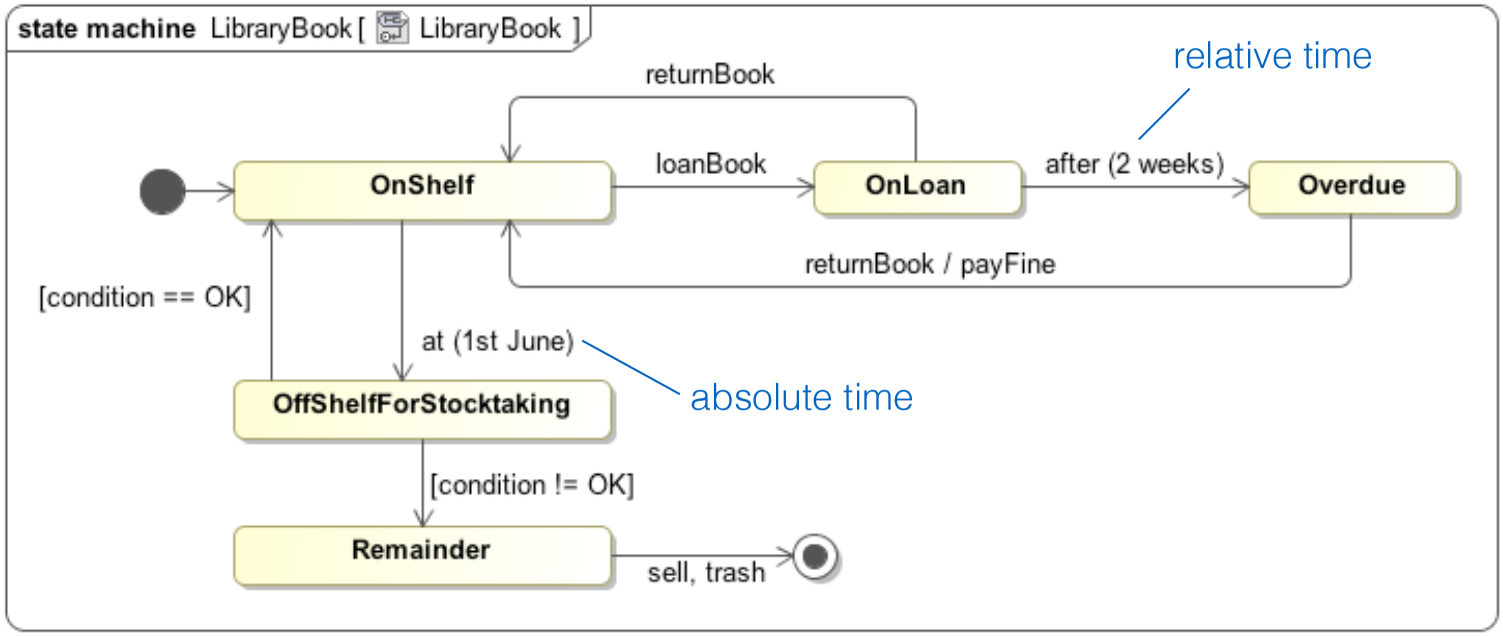
Change event





- The behavior is performed when the Boolean when(…) expression transitions from false to true. It is *edge triggered* on a false to true transition
- The values in the Boolean expression must be constants, globals or attributes of the context class
- A change event implies continually testing the condition whilst in the state

Time event



- Time events occur when a time expression becomes true. There are two types of time expression:
  - Absolute time - at(…)
  - Relative time - after(…)

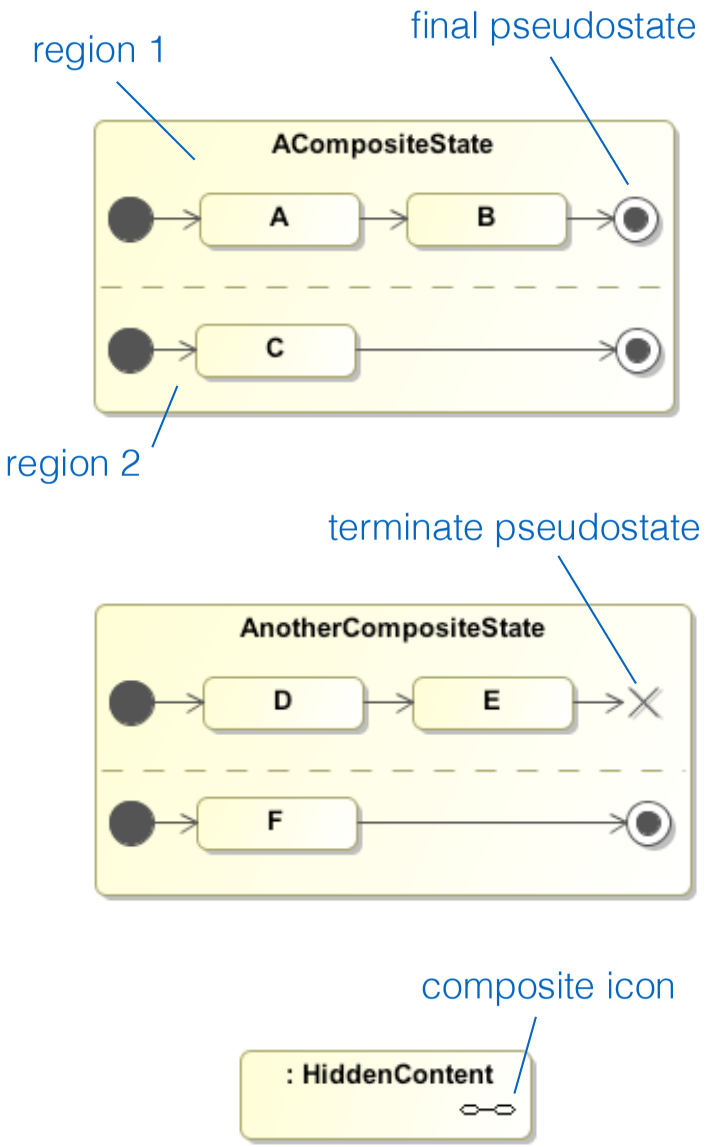
Summary

- We have looked at behavioral and protocol state machines:
- State syntax and semantics
- Behaviors including entry and exit behaviors
- Behavioral and protocol transitions
- Events: Call, signal, change and time

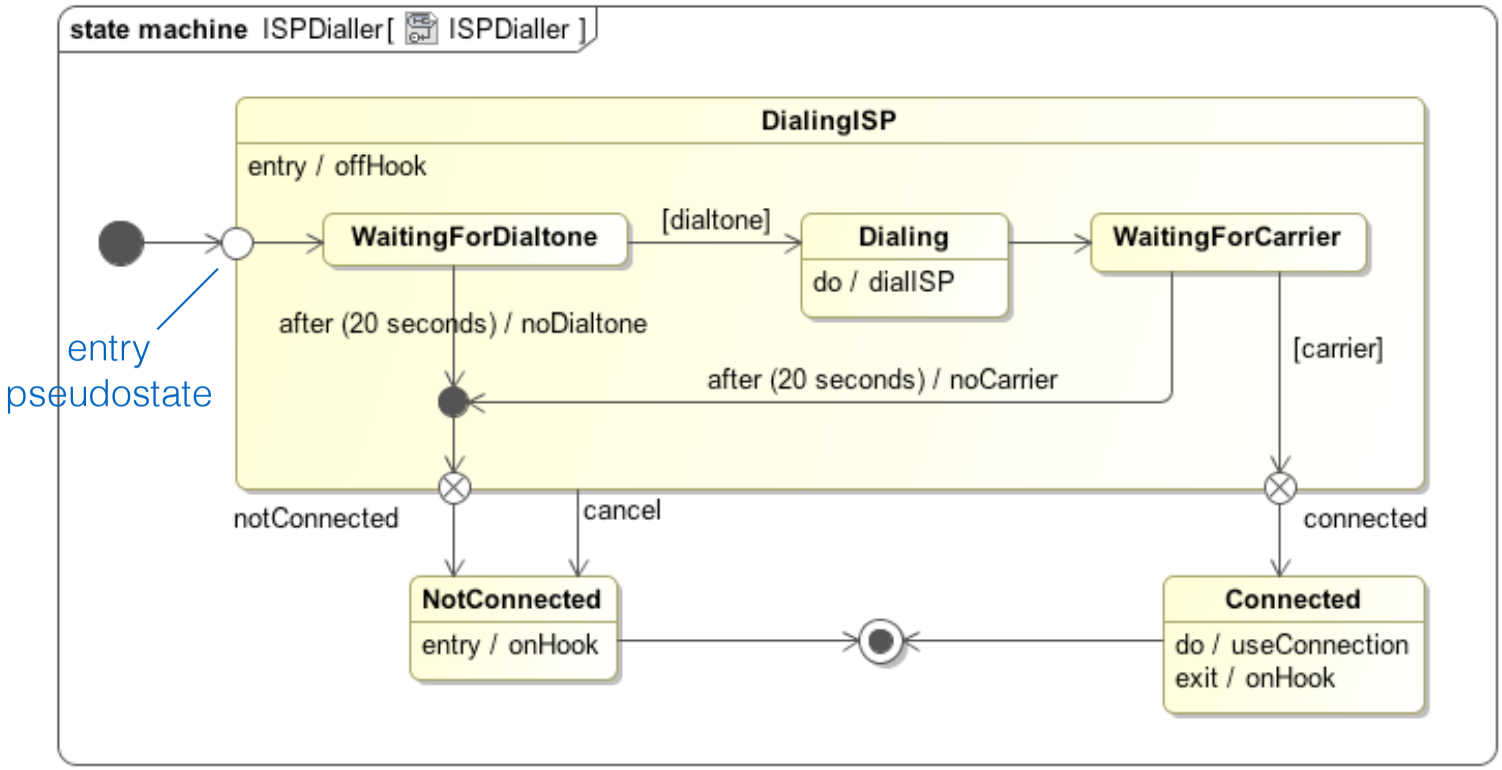
Design - advanced state machines

Composite states

- Have one or more regions that each contain a nested submachine. A *simple composite state* has exactly one region, an *orthogonal composite state* has two or more regions
- The final pseudostate terminates its enclosing region – all other regions continue to execute
- The terminate pseudostate terminates the whole state machine
- *Nested states inherit all transitions of their enclosing state*
- Use the composite icon when the contents are shown in a separate diagram

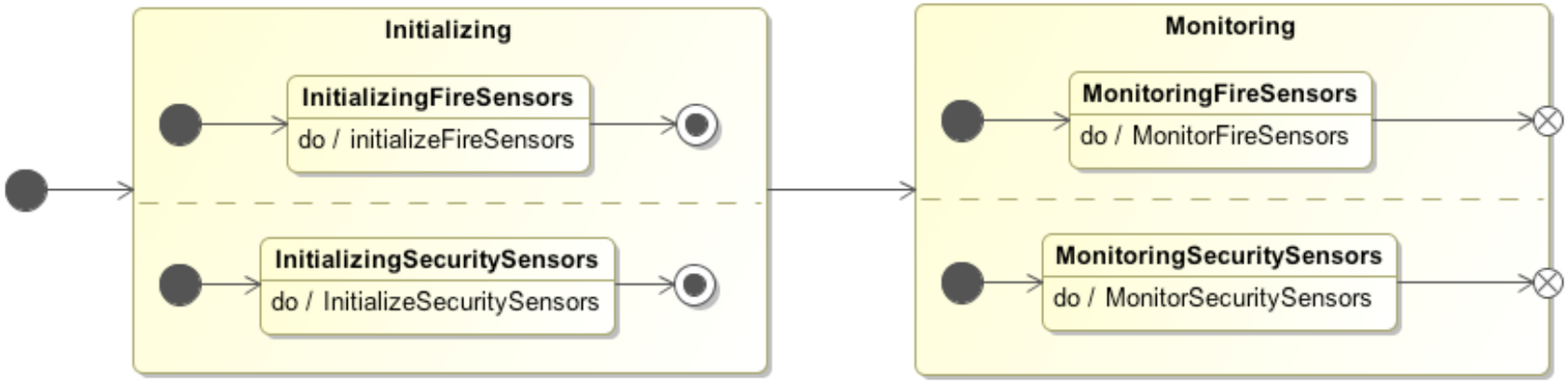


Simple composite state



- Because the nested states inherit the cancel transition from DialingISP to NotConnected - we can cancel from within any nested state

Orthogonal composite state



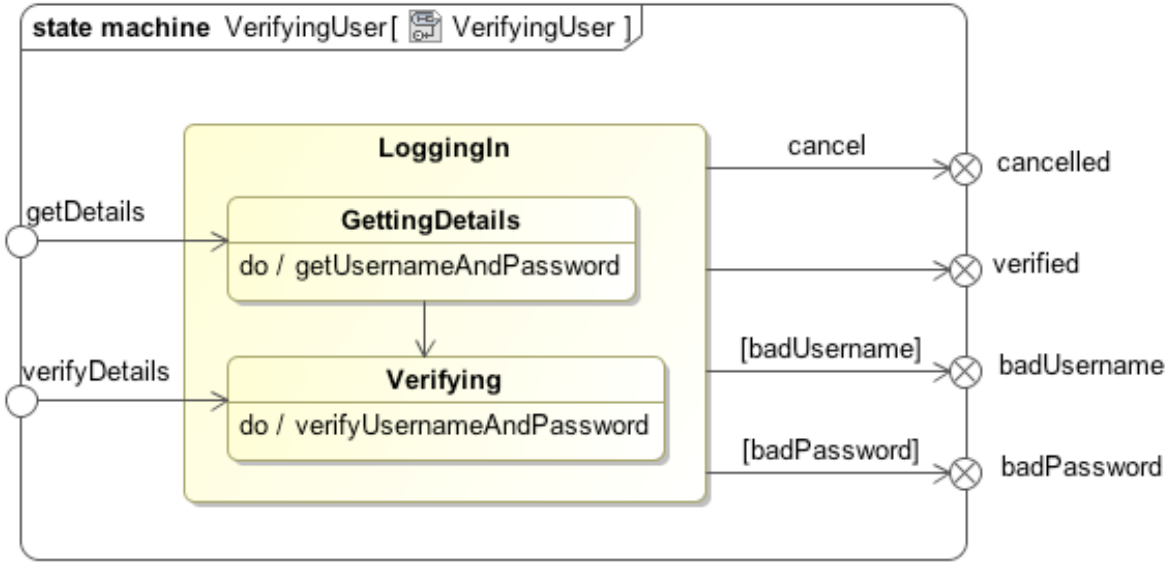
Each region can exit independently

Synchronized exit - exit the superstate when both regions have terminated

Unsynchronized exit - exit the superstate when either region terminates. The other region continues

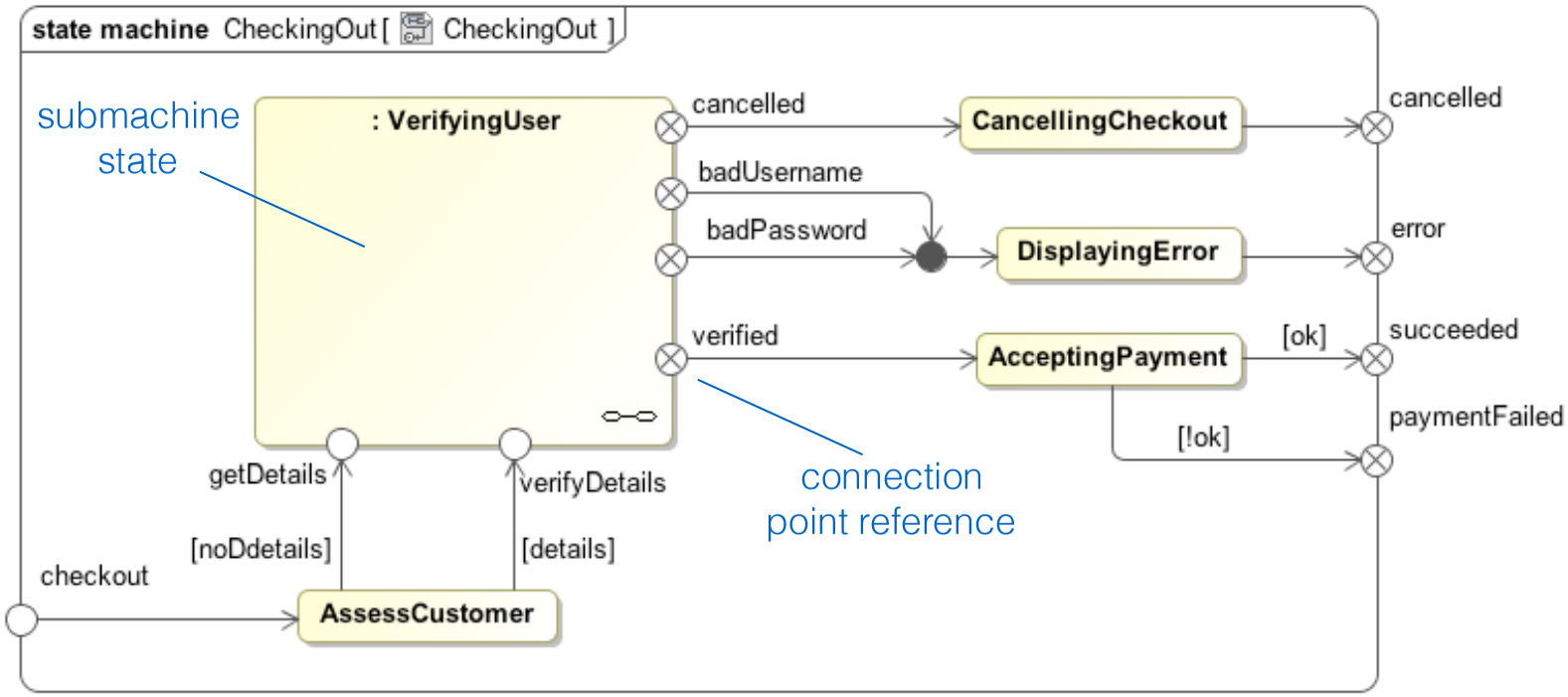
- Orthogonal composite states have two or more regions
- On entry, both submachines start executing concurrently - this is an implicit fork

Submachine state



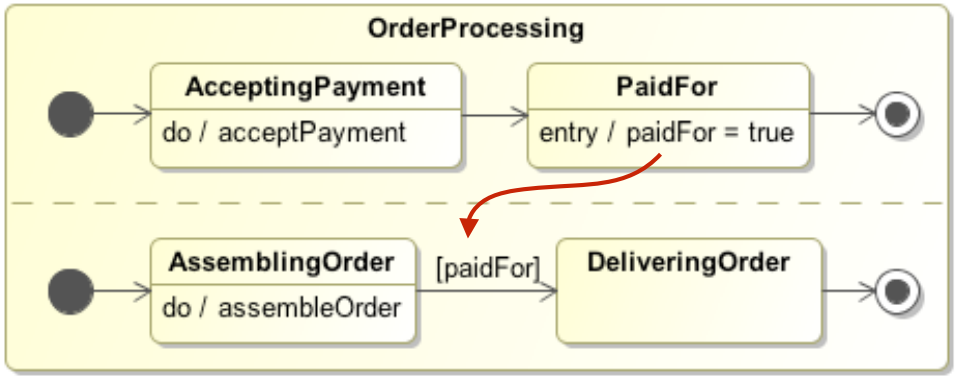
- If we want to refer to this state machine in other state machines, without cluttering the diagrams, then we must use a submachine state
- Submachine states reference another state machine
- They are similar to composite states, but reference a whole state machine even from from a different context classifier

Submachine state syntax



- A submachine state is equivalent to including a copy of the submachine in place of the submachine state
- The connection point references refer to connection points of the referenced state machine

Submachine communication



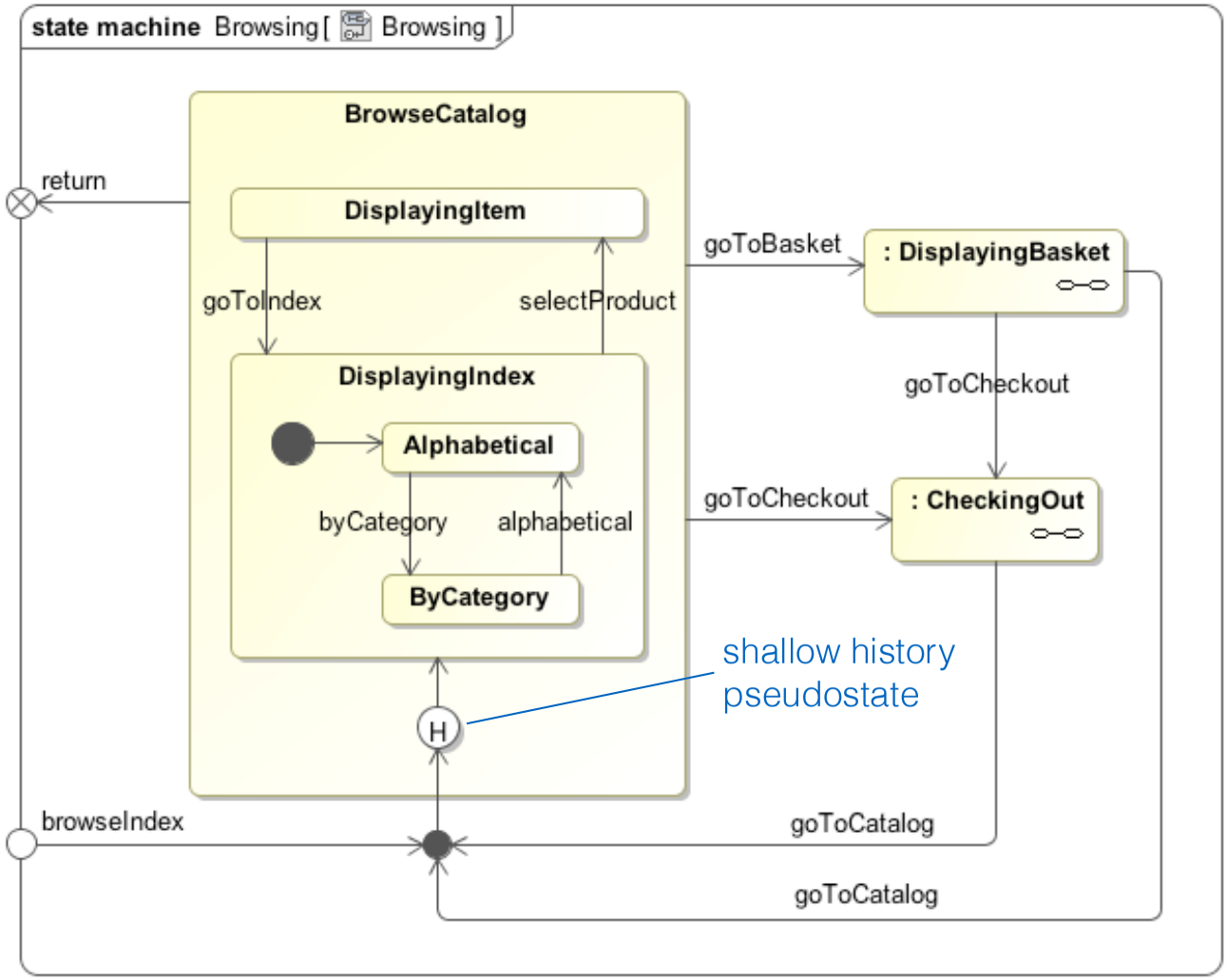
Submachine communication using the attribute paidFor as a flag.

The upper submachine sets the flag and the lower submachine uses it in a guard condition

- We often need two submachines to communicate
- Synchronous communication can be achieved by a join
- Asynchronous communication is achieved by one submachine setting a flag for another one to process in its own time
- Use attributes of the context object as flags

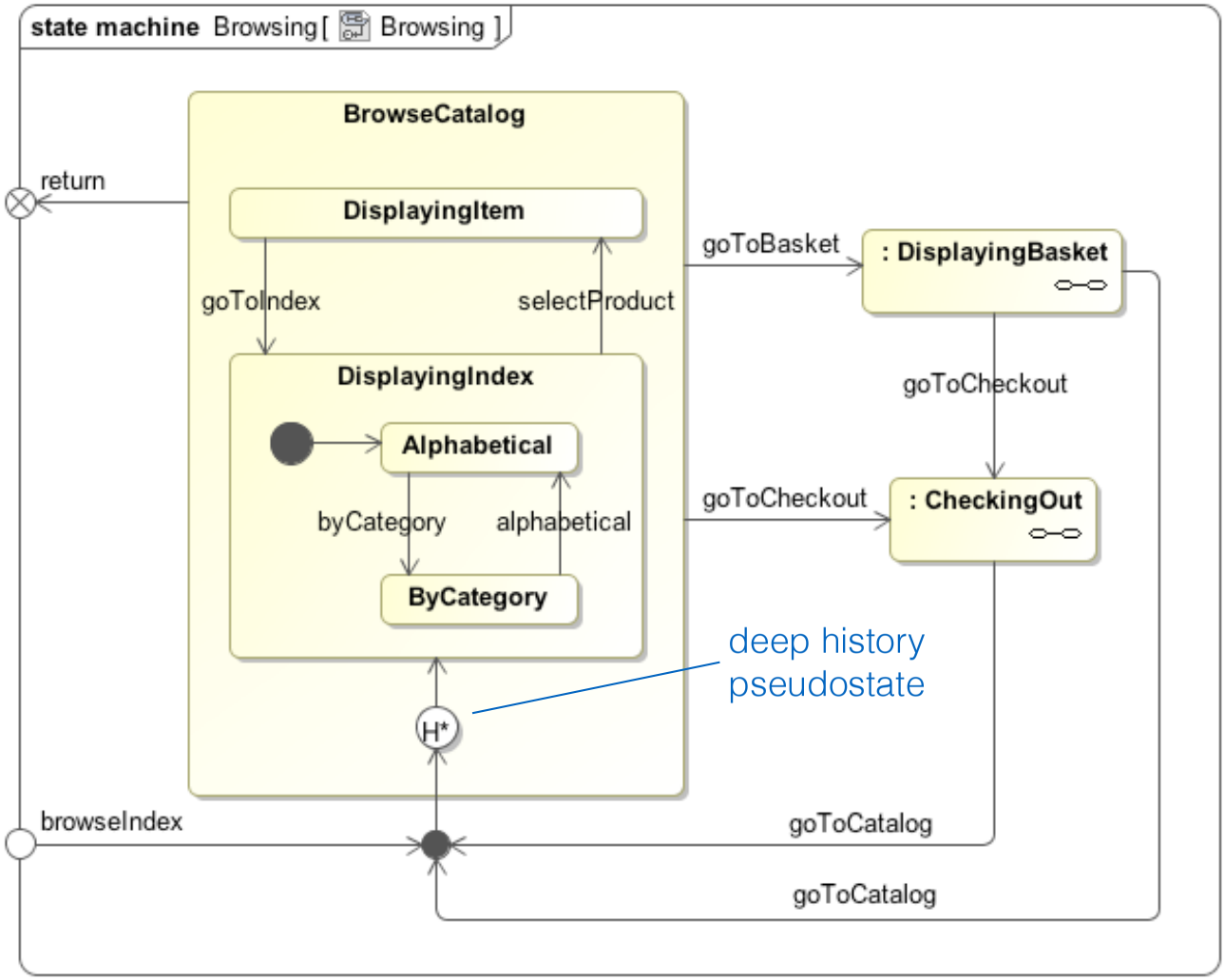
Shallow history

- The shallow history pseudostate remembers the last substate *at the same level as itself*, i.e. DisplayingItem and Displayingindex
- The shallow history pseudostate fires the *first* time the history state is entered. The *next* time there is an automatic transition to the remembered substate



Deep history

- The deep history pseudostate remembers the last substate *at the same level or lower* than itself, i.e. DisplayingItem, DisplayingIndex, Alphabetical and ByCategory



Summary

- We have explored more advanced aspects of state machines including:
  - Simple composite states
  - Orthogonal composite states
  - Submachine communication using attribute values
  - Submachine states
  - Shallow history and deep history