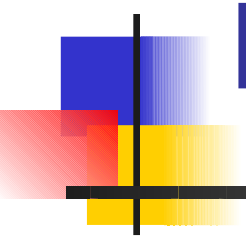
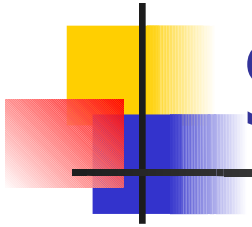


Design - state machines

© Clear View Training 2008 v2.5

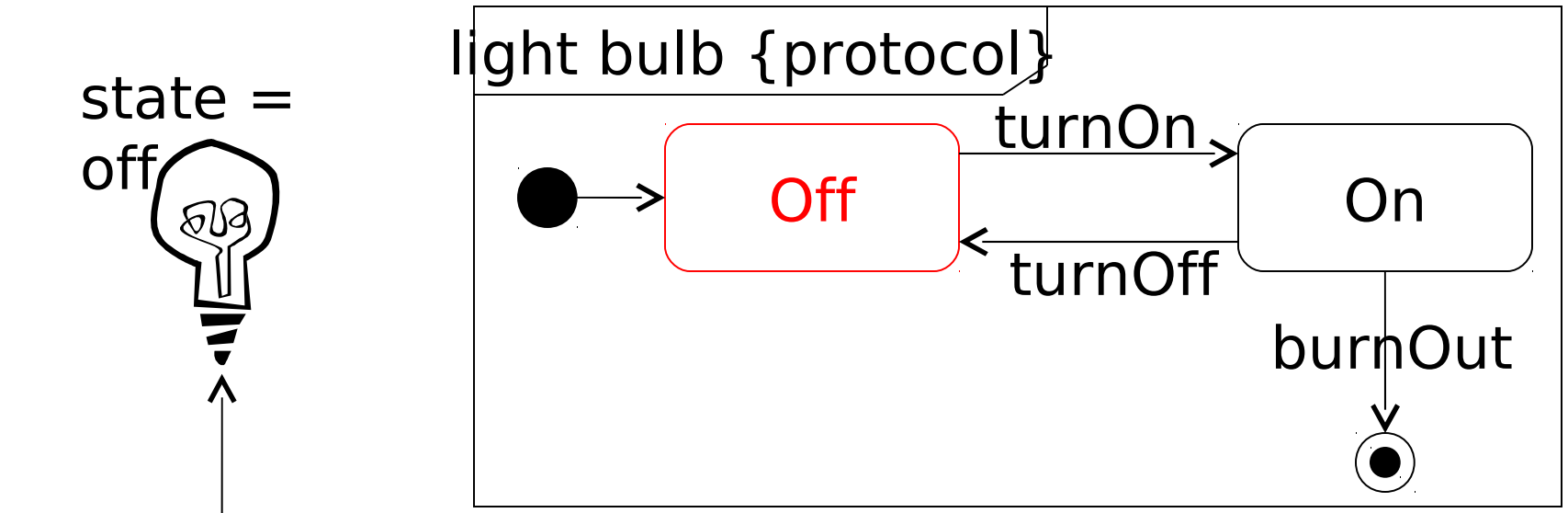


State machines

- Some model elements such as classes, use cases and subsystems, can have interesting dynamic behavior - state machines can be used to model this behaviour
- 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 modelled as a progression of *states, transitions* and *events.* We'll see what these mean in a minute!
  - Its current behaviour depends on its past
- A state machine diagram always contains exactly one state machine for one model element
- There are two types of state machines (see next slide):
  - *Behavioural* state machines - define the behavior of a model element e.g. the behavior of class instances
  - *Protocol* state machines - Model the protocol of a classifier
    - The conditions under which operations of the classifier can be called
    - The ordering and results of operation calls
    - Can model the protocol of classifiers that have no behavior (e.g. interfaces and ports)

© Clear View Training 2008 v2.5

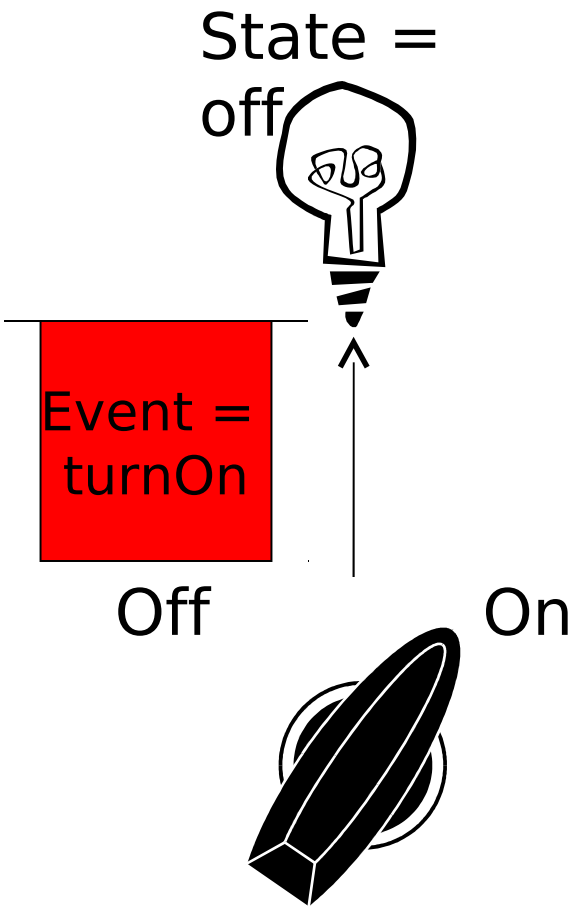
State machine diagrams

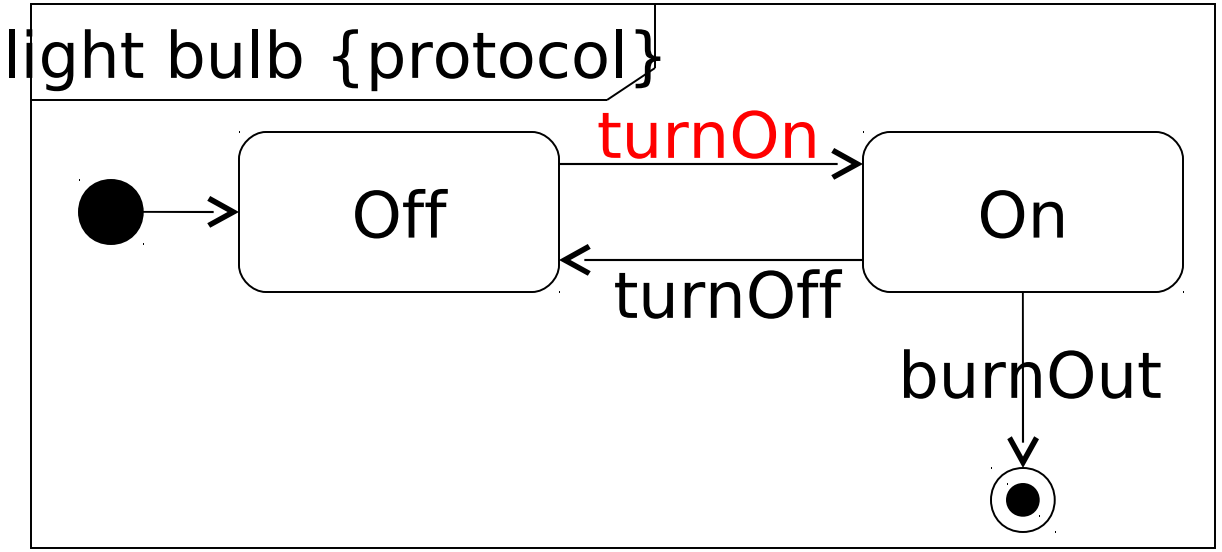


- We begin with the light bulb in the state off

© Clear View Training 2008 v2.5

Light bulb turnOn

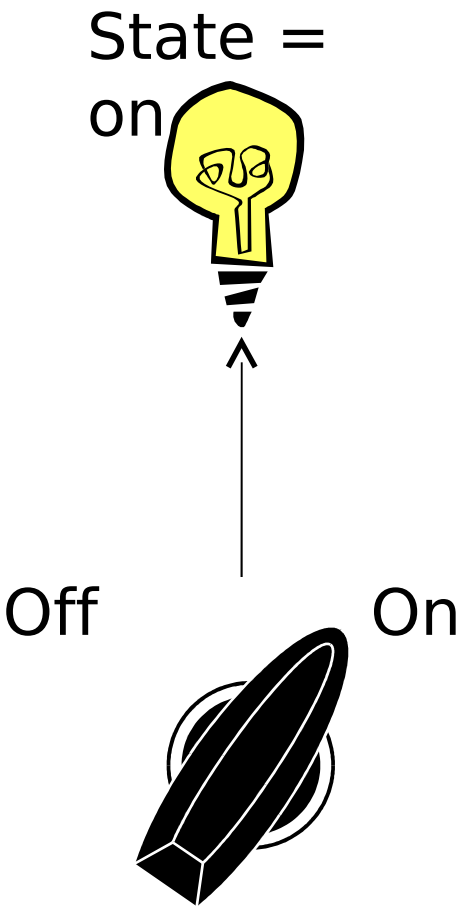


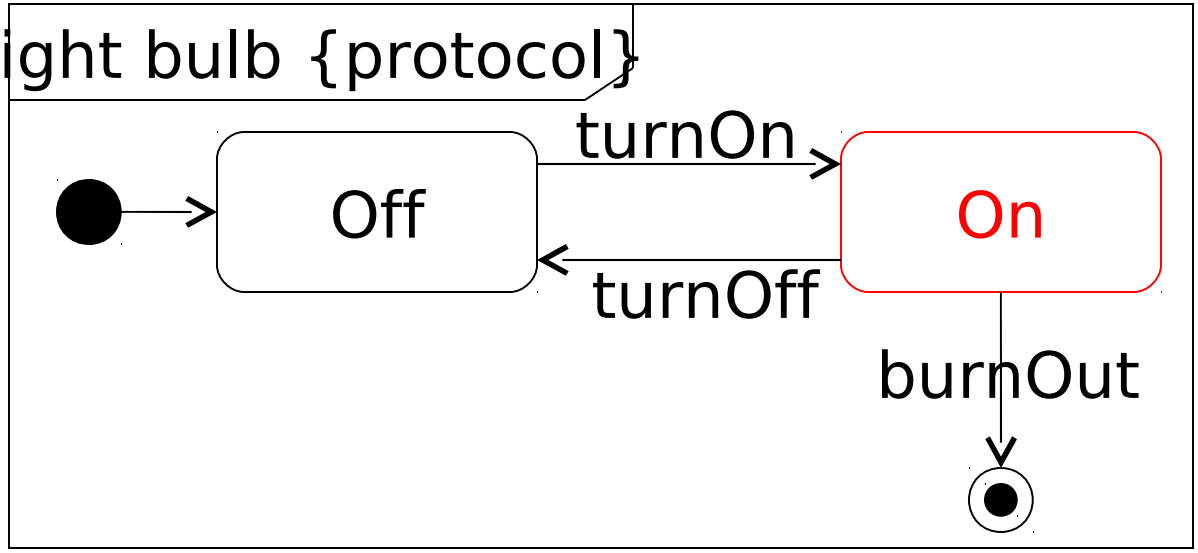


- We throw the switch to On and the event turnOn is sent to the lightbulb

© Clear View Training 2008 v2.5

Light bulb On

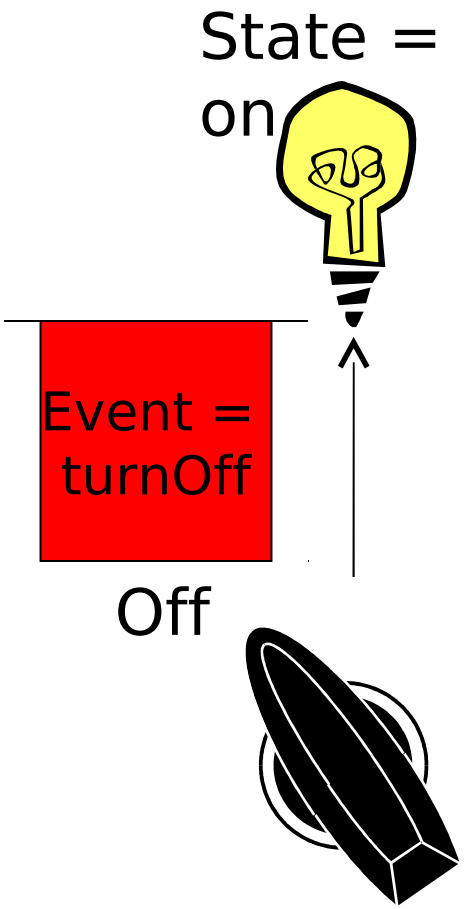


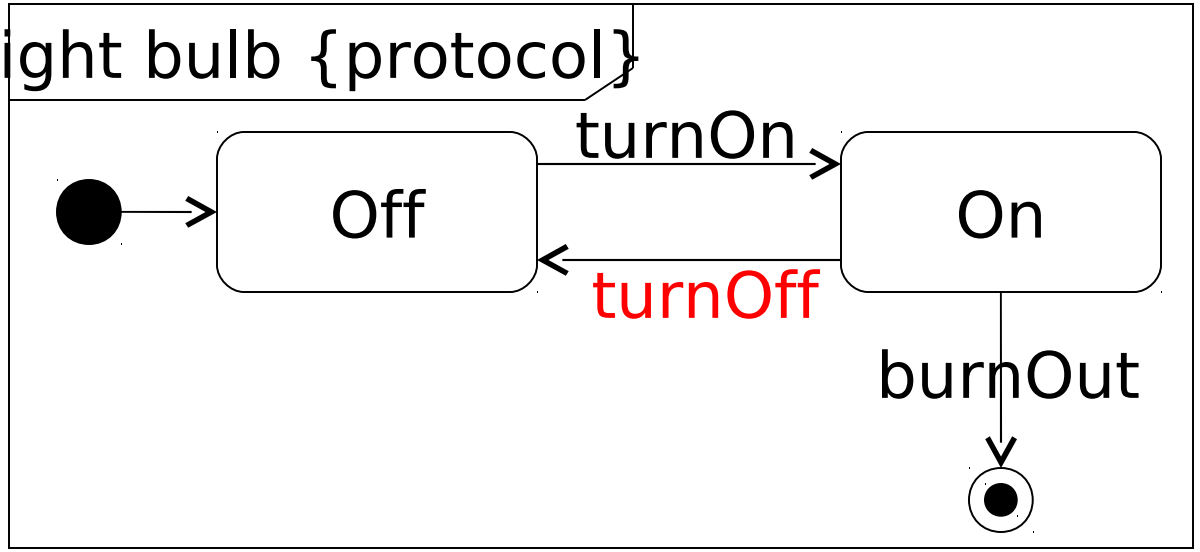


- The light bulb turns on

© Clear View Training 2008 v2.5

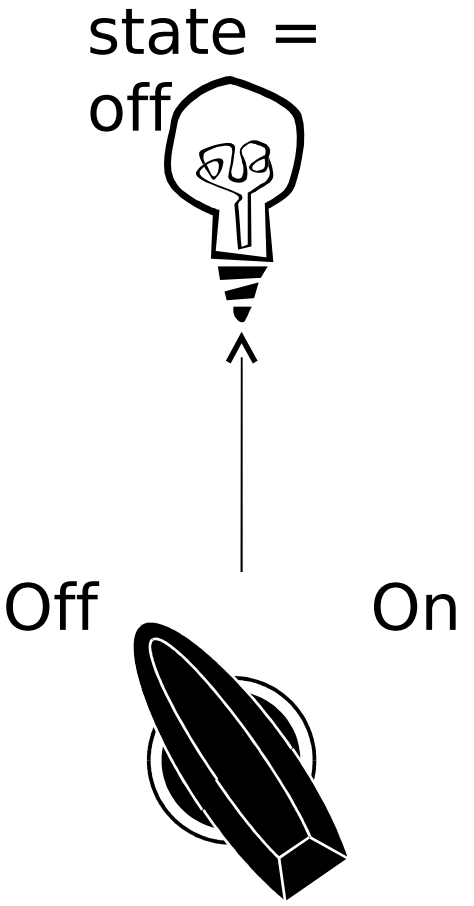
Light bulb turnOff

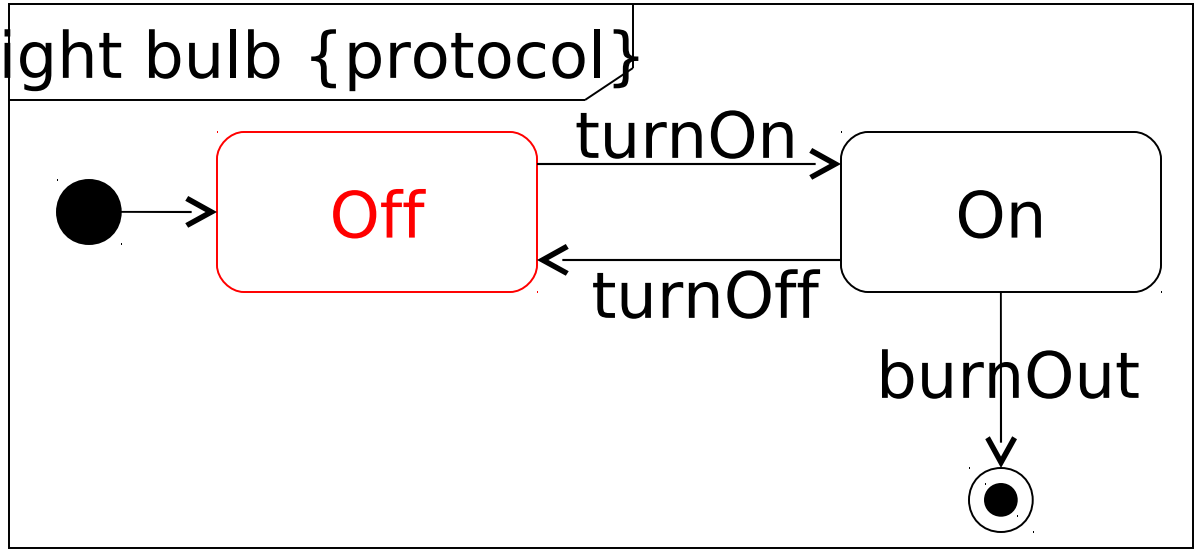




- We turn the switch to Off. The event turnOff is sent to the light bulb

Light bulb Off

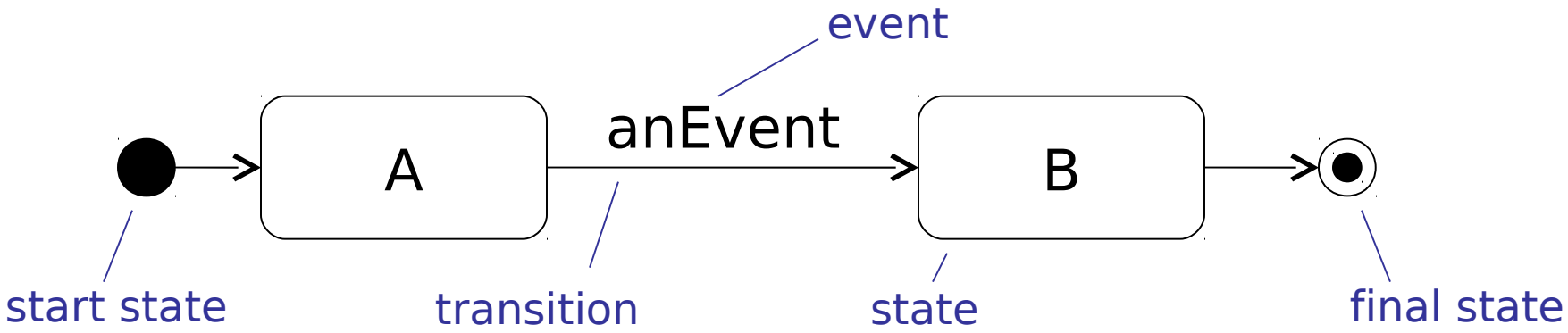




- The light bulb turns off

© Clear View Training 2008 v2.5

Basic state machine syntax



- Every state machine should have a start state 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!

States

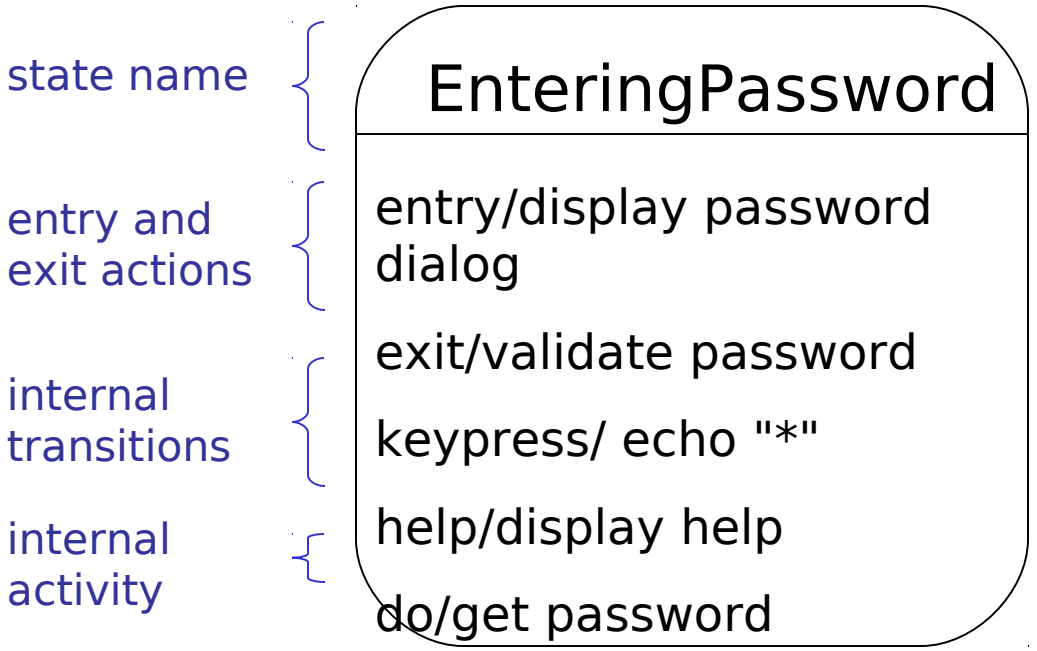
- "A condition or situation during the life of an object during which it satisfies some condition, performs some activity or waits for some event"
- The state of an object at any point in time is determined by:
  - The values of its attributes
  - The relationships it has to other objects
  - The activities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

How many states?

Color
red : int green : int blue : int

State syntax

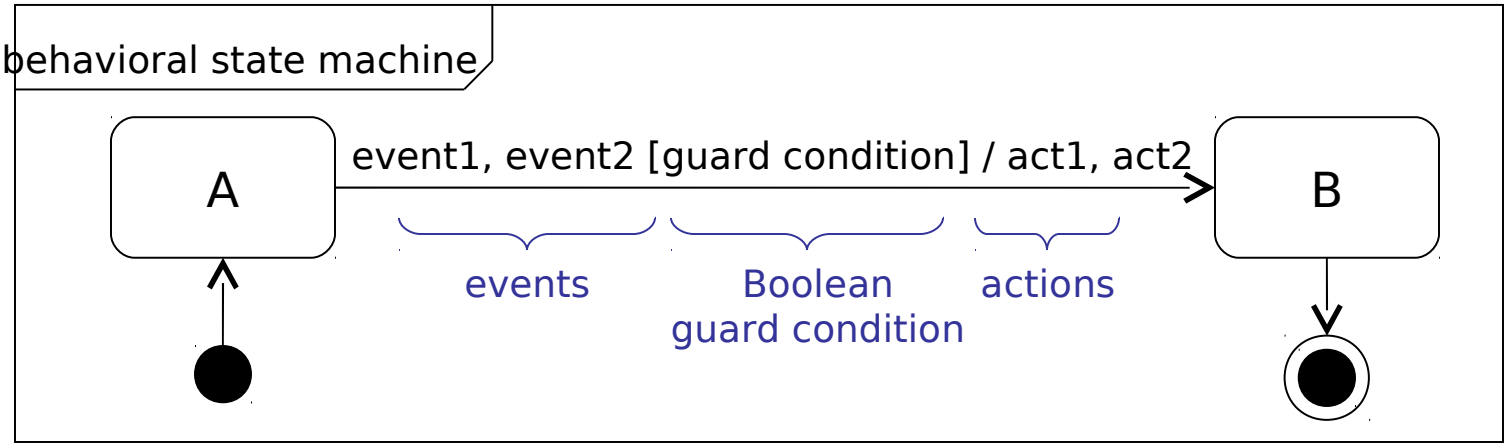
- Actions are *instantaneous* and *uninterruptible*
  - Entry actions occur immediately on entry to the state
  - Exit actions occur immediately on leaving the state
- Internal transitions occur *within* the state. They do *not* transition to a new state
- Activities take a finite amount of time and are interruptible



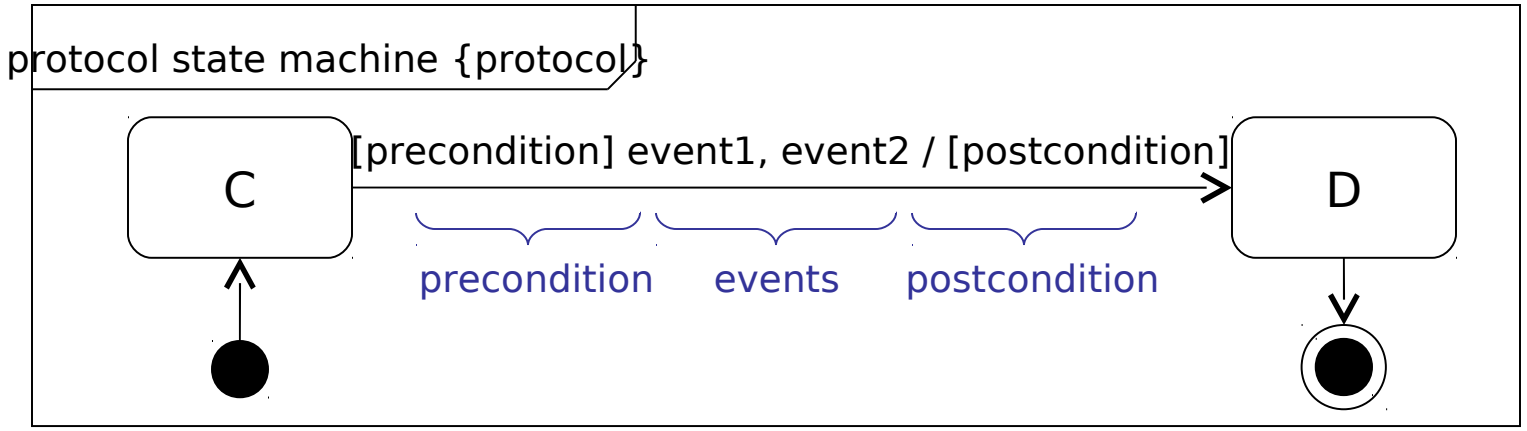
Action syntax: eventTrigger / action
Activity syntax: do / activity

Transitions

behavioral state machine



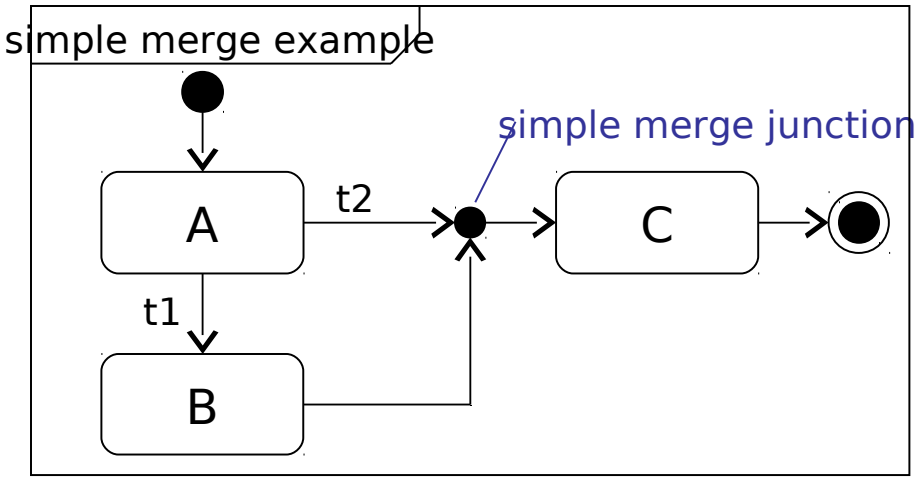
protocol state machine

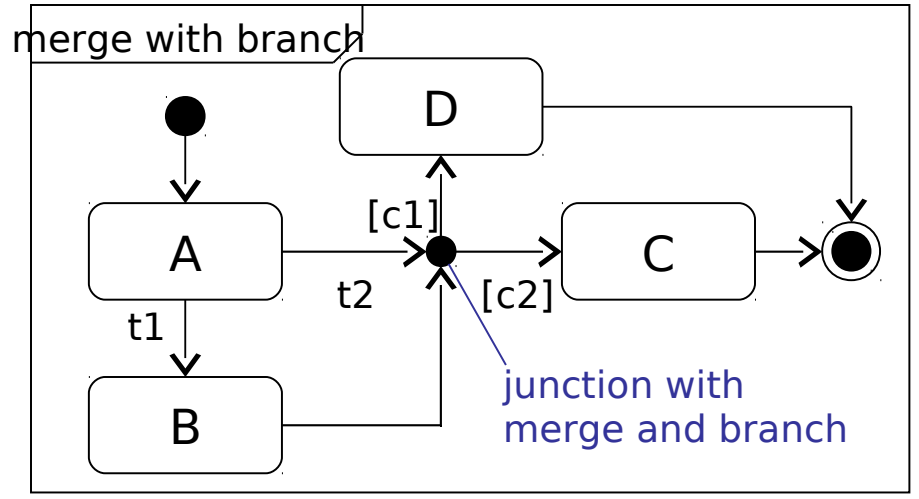


© Clear View Training 2008 v2.5

Connecting - the junction pseudo state

- The junction pseudo state can:
  - connect transitions together (merge)
  - branch transitions
- Each outgoing transition must have a mutually exclusive guard condition

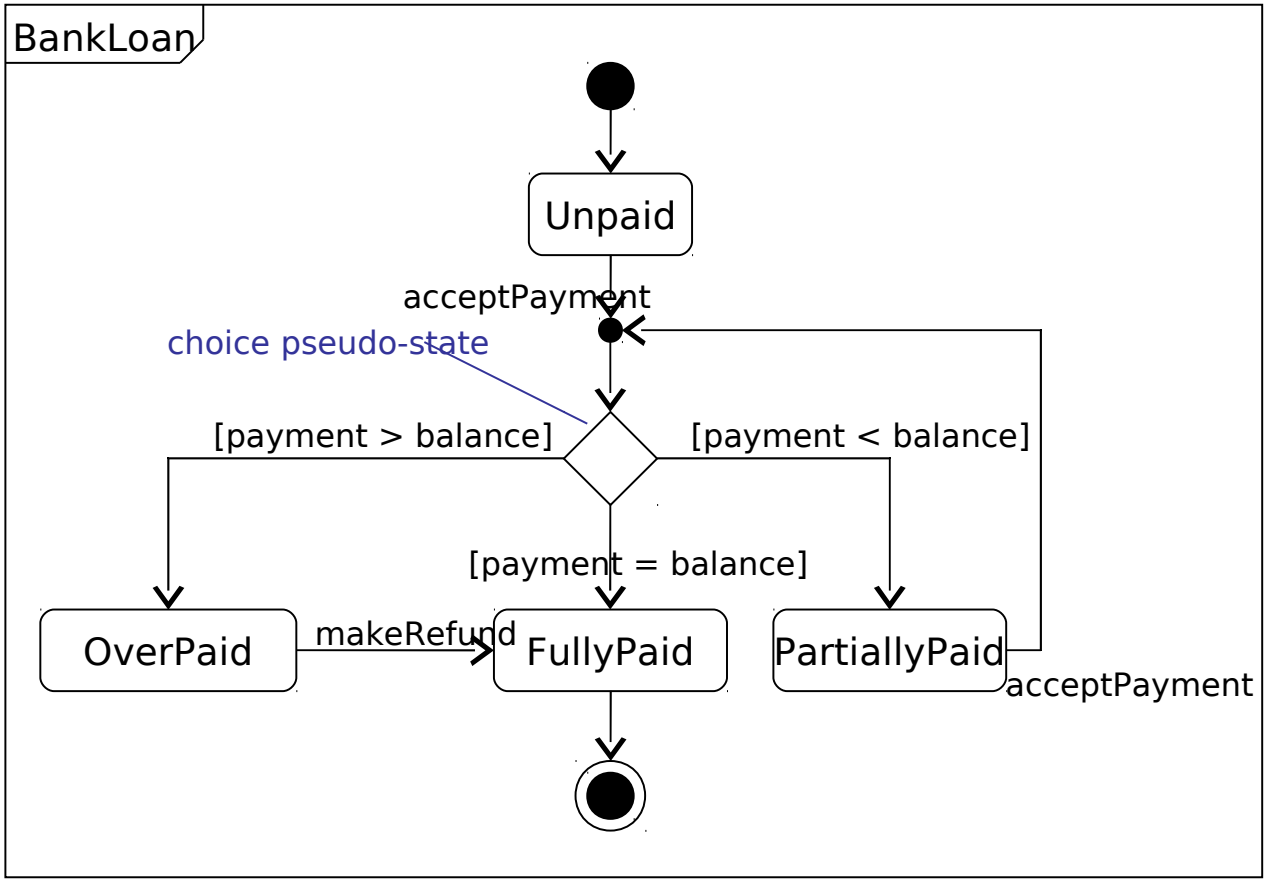




© Clear View Training 2008 v2.5

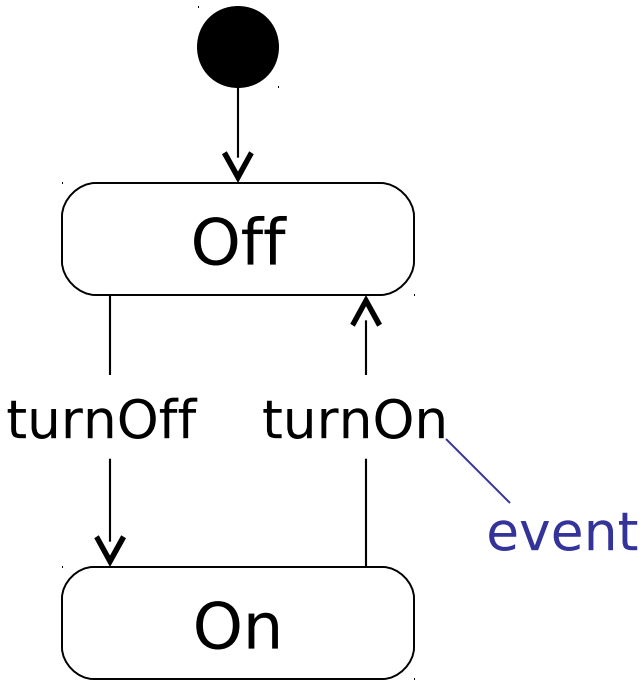
Branching – the choice pseudo state

- The choice pseudo state directs its single incoming transition to one of its outgoing transitions
- Each outgoing transition must have a mutually exclusive guard condition



Events

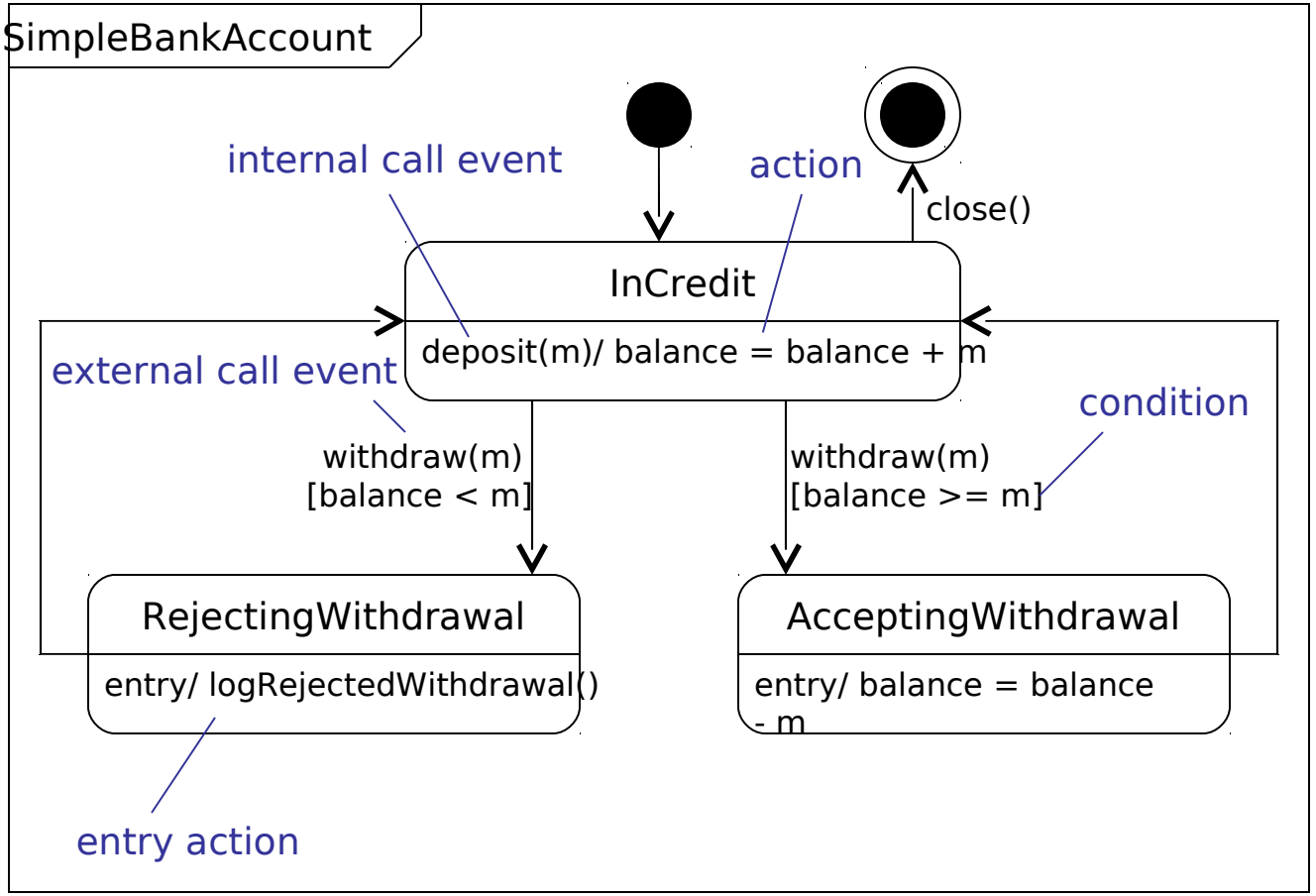
- "The specification of a noteworthy occurrence that has location in time and space"
- Events trigger transitions in state machines
- Events can be shown externally, on transitions, or internally within states (internal transitions)
- There are four types of event:
  - Call event
  - Signal event
  -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



© Clear View Training 2008 v2.5

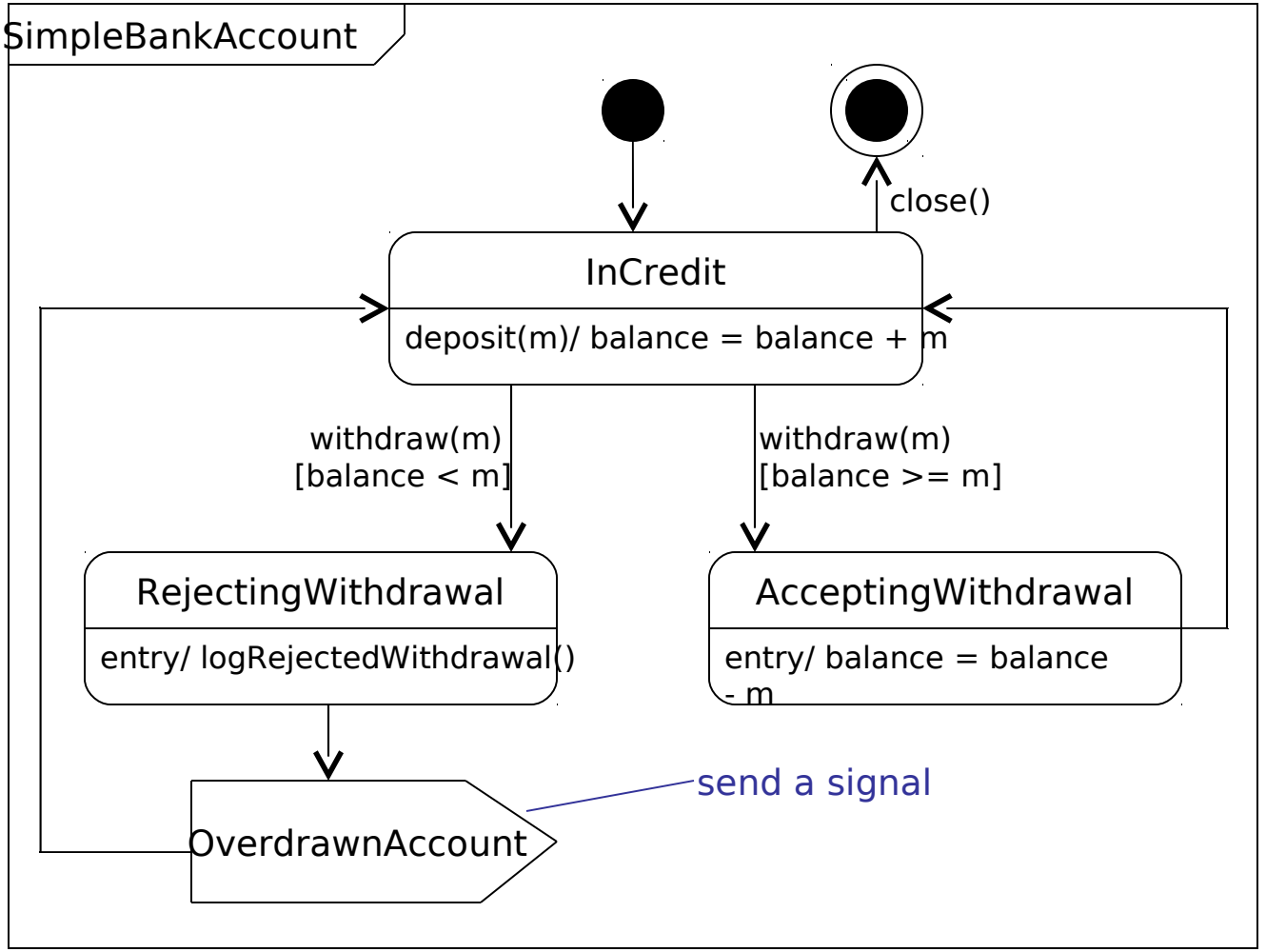
Call event

- A call for an operation executon
- The event should have the same signature as an operation of the context class
- A sequence of actions may be specified for a call event - they may use attributes and operations of the context class
- The return value must match the return type of the operation



Signal events

- A signal is a package of information that is sent asynchronously between objects
  - the attributes carry the information
  - no operations

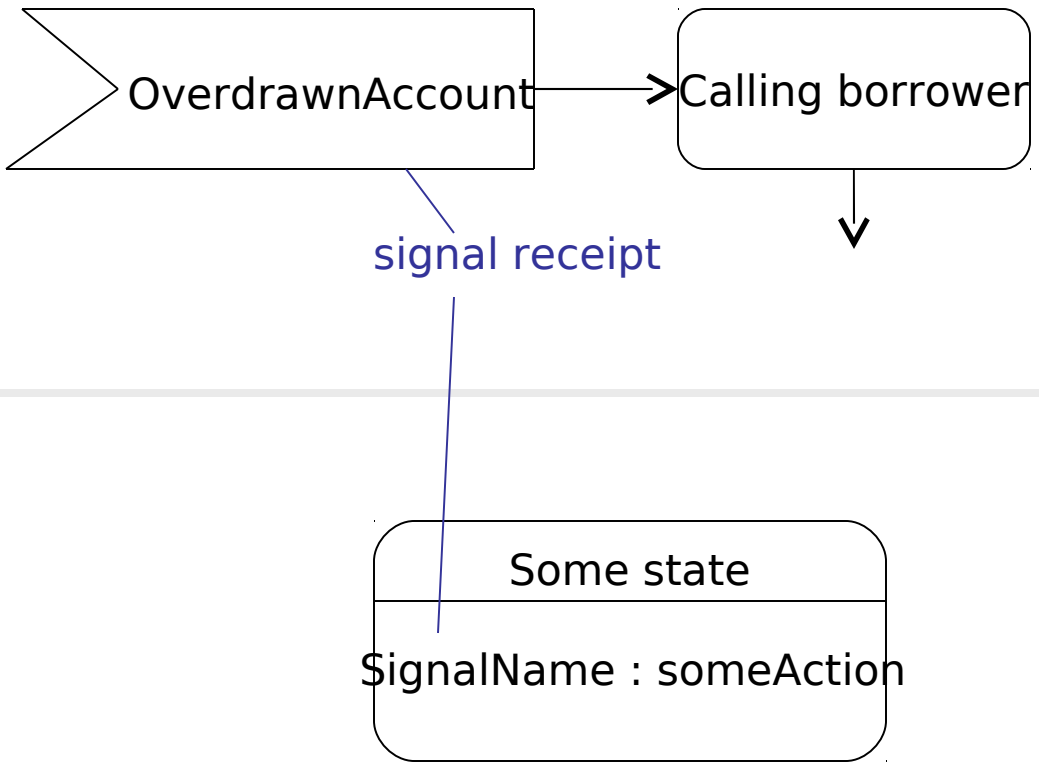


«signal»
OverdrawnAccount

date : Date
accountNumber : long
amountOverdrawn : double

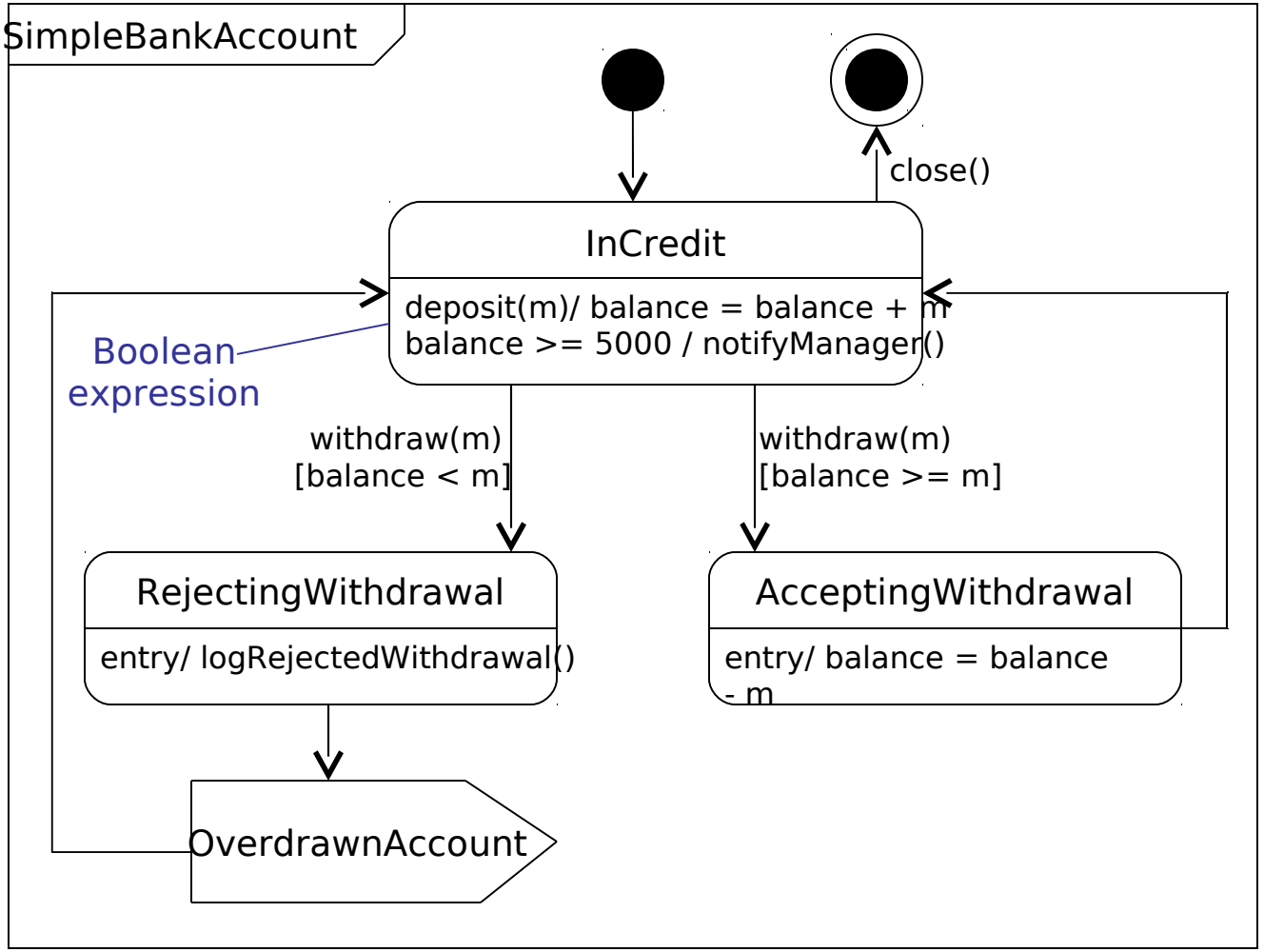
Receiving a signal

- You may show a signal receipt on a transition using a concave pentagon or as an internal transition state using standard notation



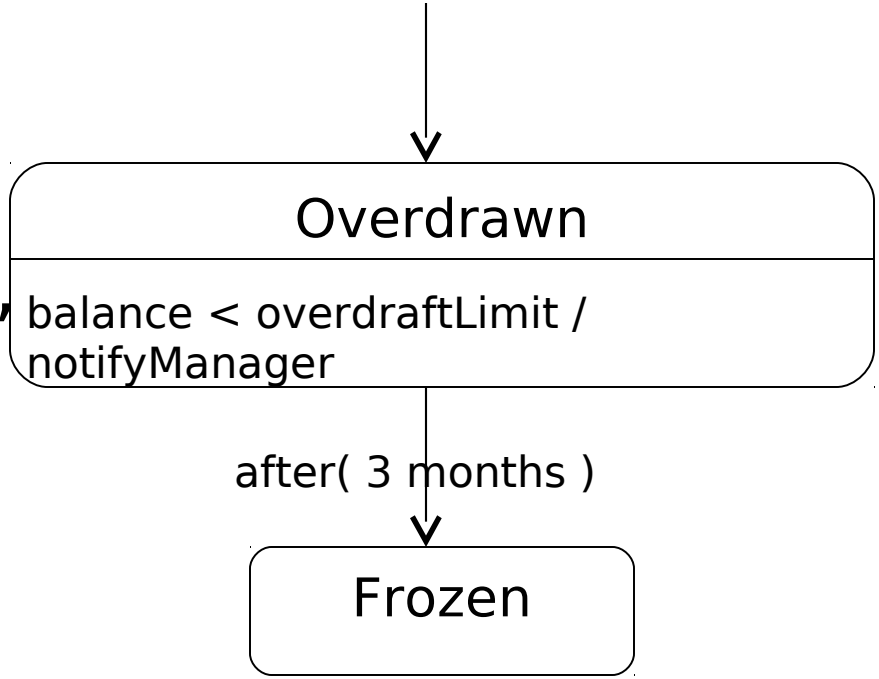
Change events

- The action is performed when the Boolean expression transitions from false to true
  - The event 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



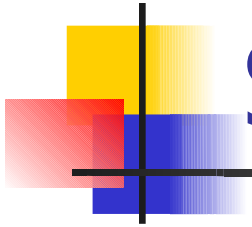
Time events

- Time events occur when a time expression becomes true
- There are two keywords, after and when
- Elapsed time:
  - after(3 months)
- Absolute time:
  - when(date =20/3/2000)



Context: CreditAccount class

© Clear View Training 2008 v2.5



Summary

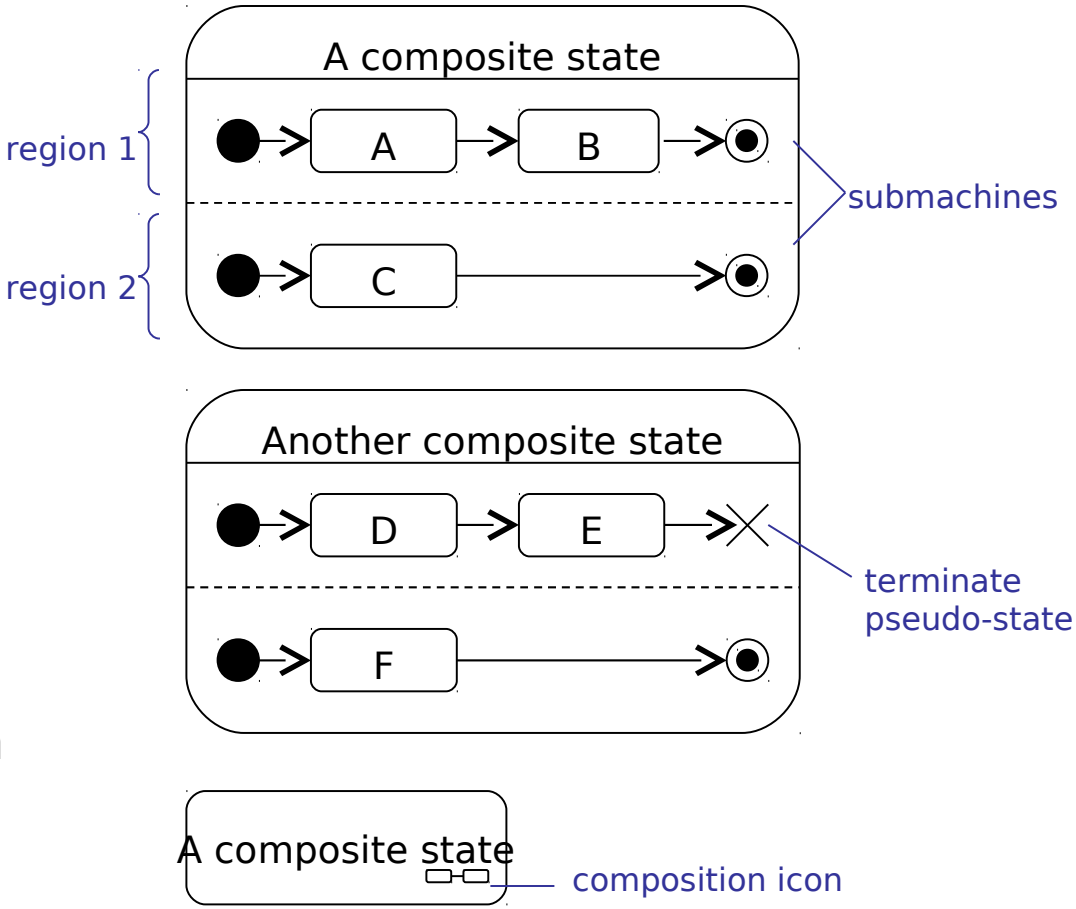
- We have looked at:
  - Behavioral state machines
  - Protocol state machines
  - States
    - Actions
      - Exit and entry actions
    - Activities
  - Transitions
    - Guard conditions
    - Actions
  - Events
    - Call, signal, change and time

© Clear View Training 2008 v2.5

Design - advanced state machines

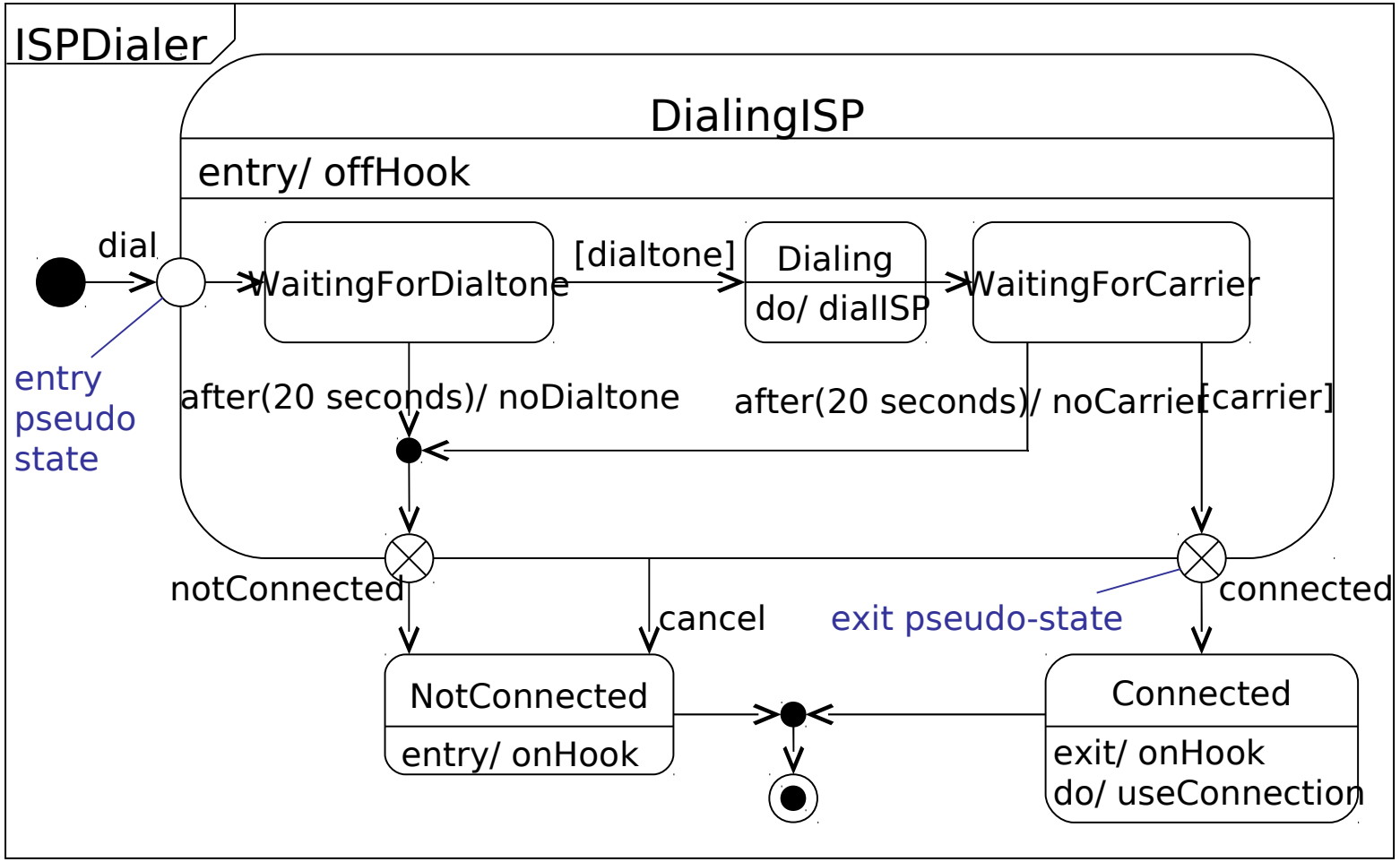
Composite states

- Have one or more regions that each contain a nested submachine
  - Simple composite state
    - exactly one region
  - Orthogonal composite state
    - two or more regions
- The final state terminates its enclosing region – all other regions continue to execute
- The terminate pseudo-state terminates the whole state machine
- Use the composition icon when the submachines are hidden



Simple composite states

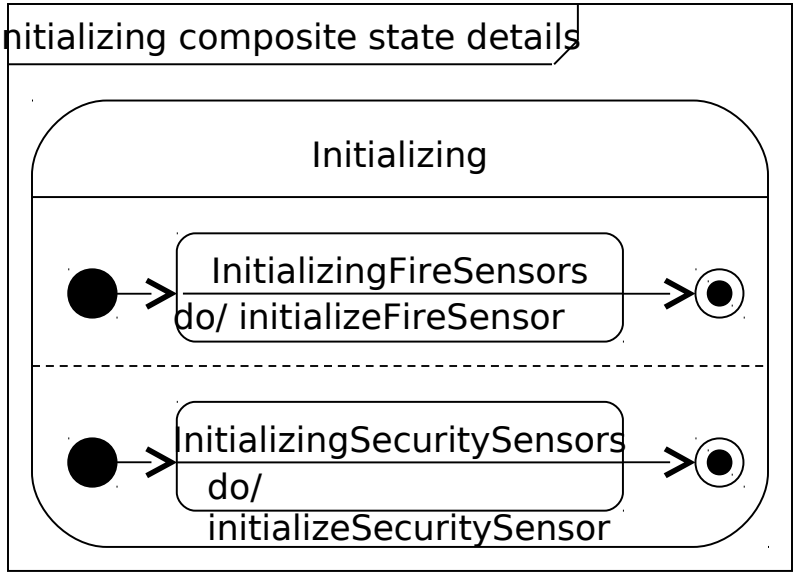
- Contains a single region



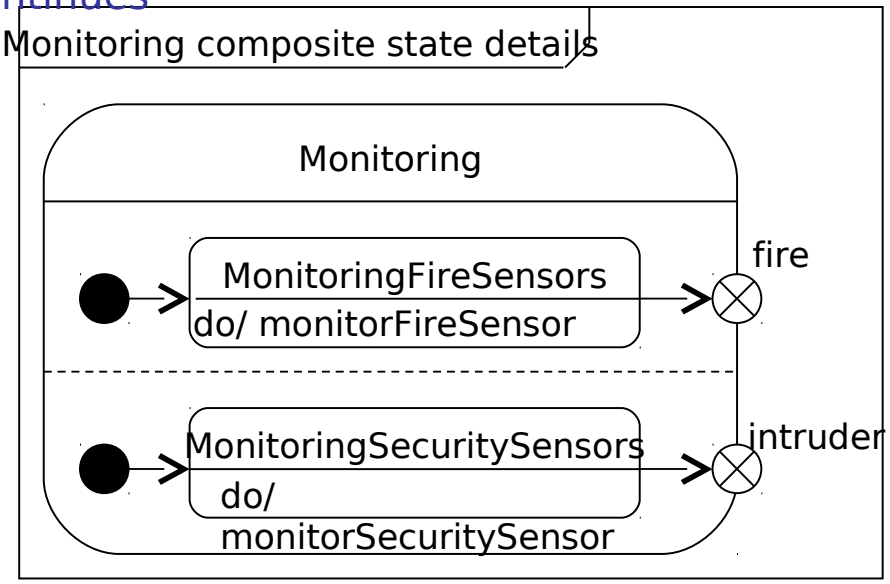
Orthogonal composite states

- Has two or more regions
- When we enter the superstate, both submachines start executing concurrently - this is an implicit fork

Synchronized exit - exit the superstate when *both* regions have terminated

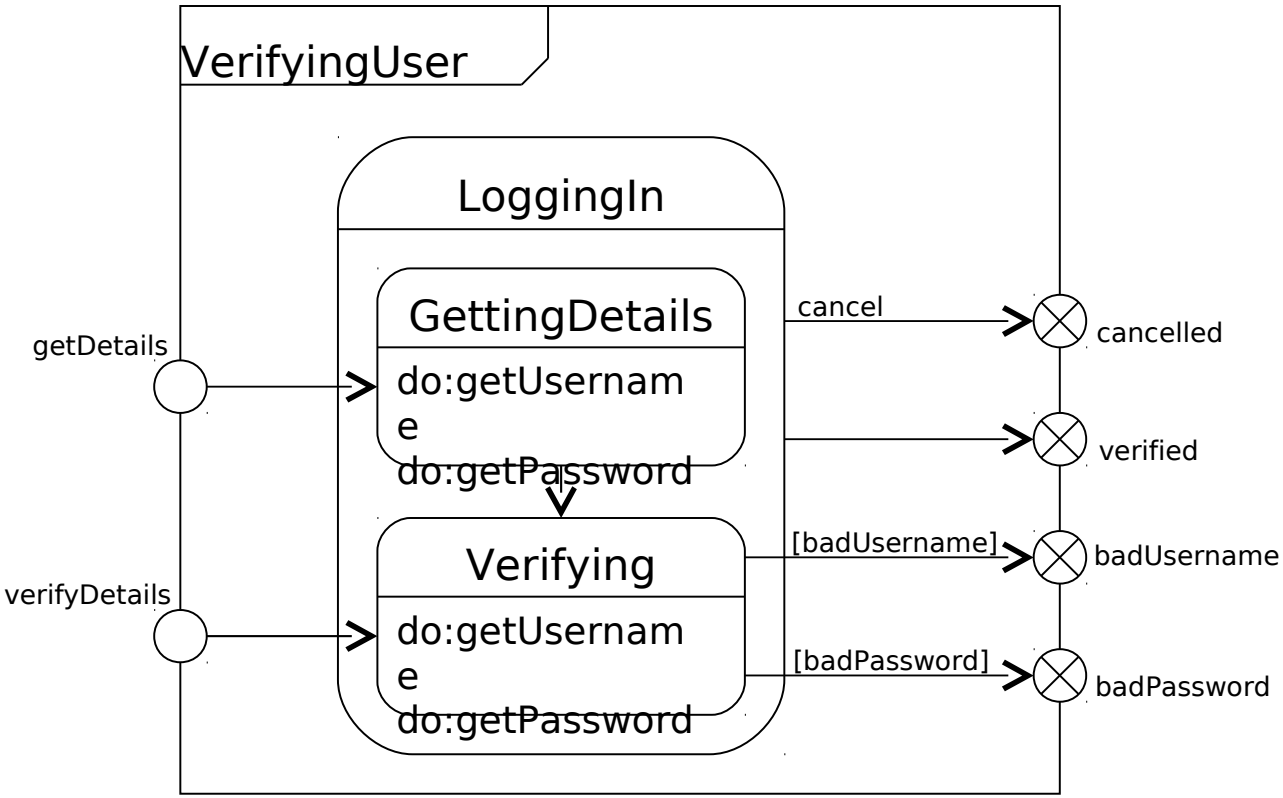


Unsynchronized exit - exit the superstate when *either* region terminates. The other region continues



Submachine states

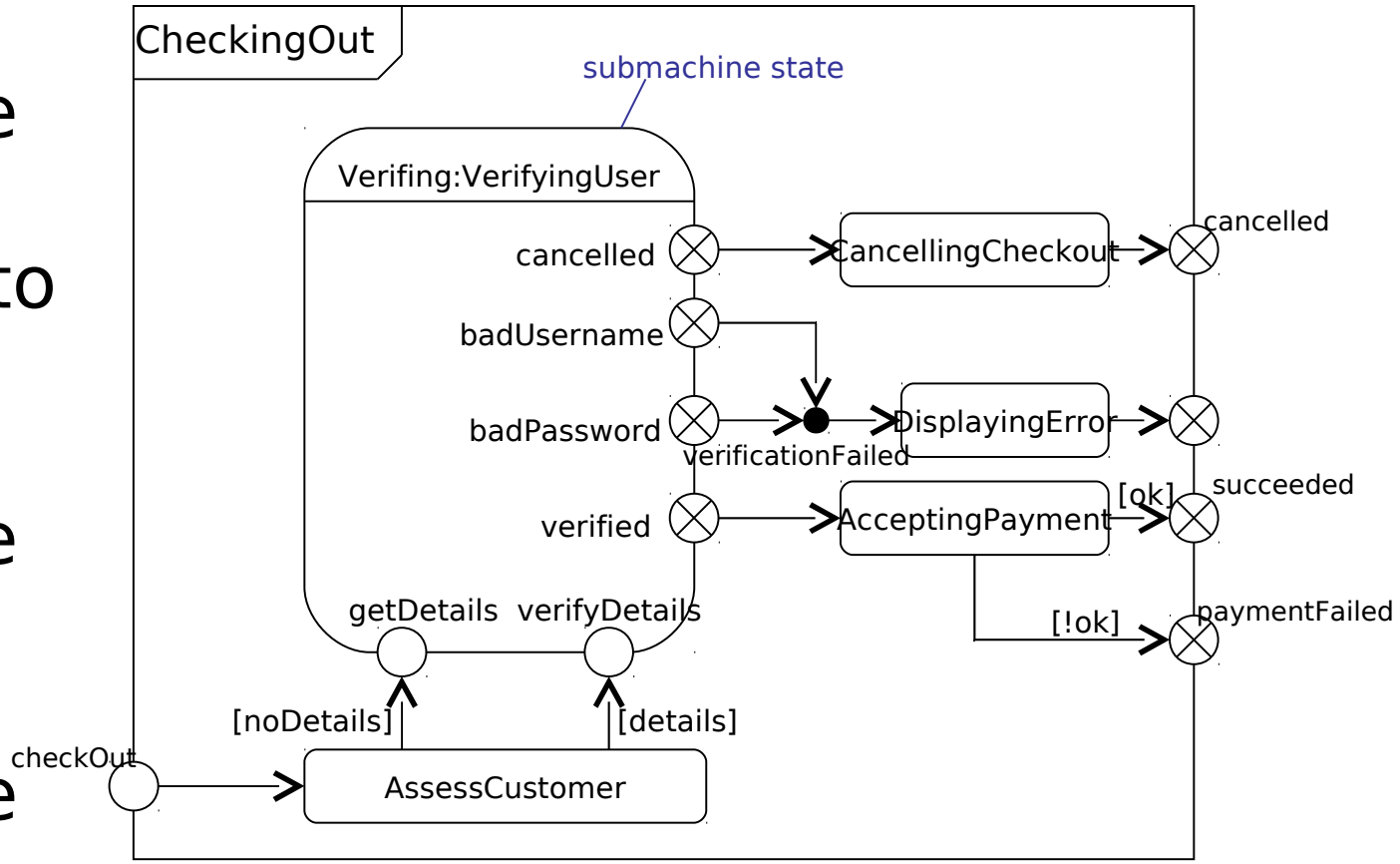
- If we want to refer to this state machine in other state machines, without cluttering the diagrams, then we must use a *submachine state*
- Submachine states reference another state machine
- Submachine states are semantically equivalent to composite states



© Clear View Training 2008 v2.5

Submachine state syntax

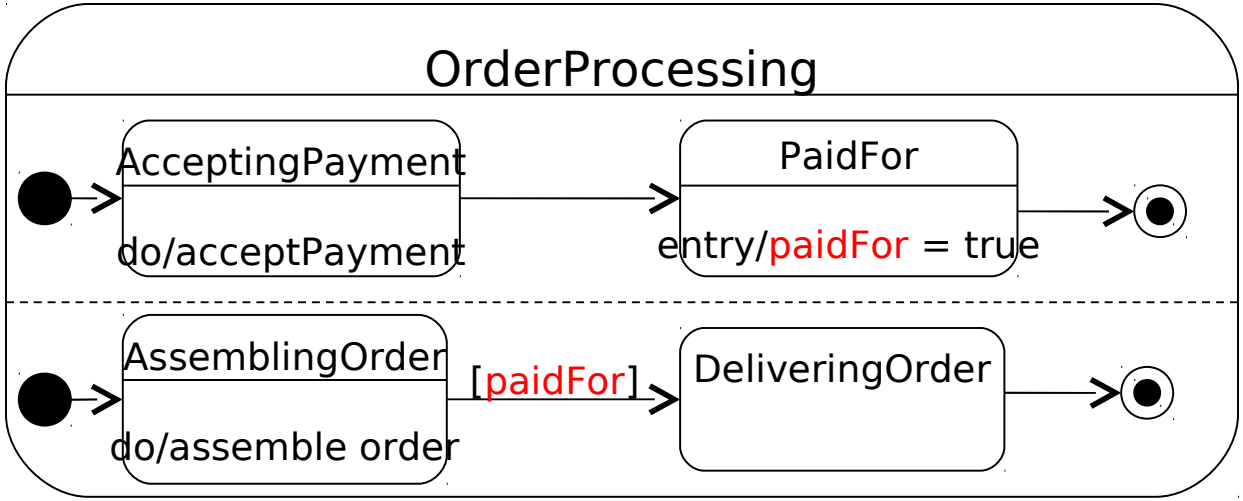
- A submachine state is equivalent to including a copy of the submachine in place of the submachine state

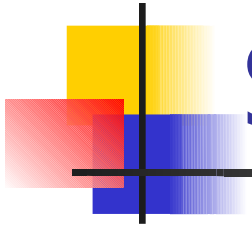


Submachine communication

- 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

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





Summary

- We have explored advanced aspects of state machines including:
  - Simple composite states
  - Orthogonal composite states
  - Submachine communication
    - Attribute values
  - Submachine states

© Clear View Training 2008 v2.5