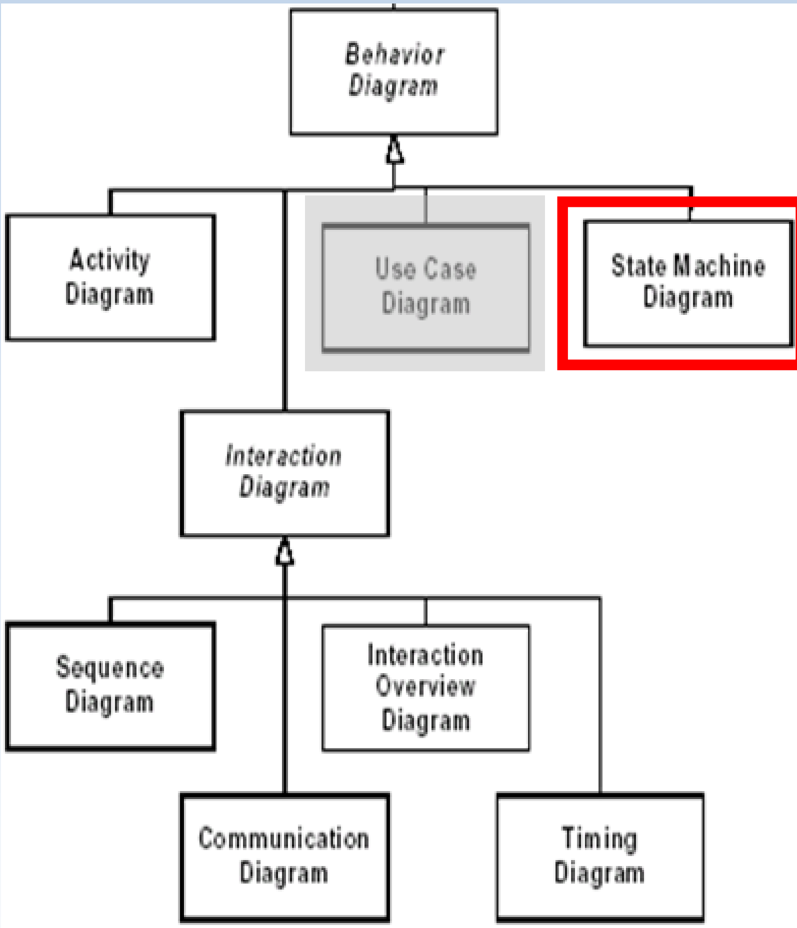
DEI
DEPARTAMENTO
DE ENGENHARIA INFORMÁTICA
TÉCNICO LISBOA

Behavioural Modelling (with UML and SysML)

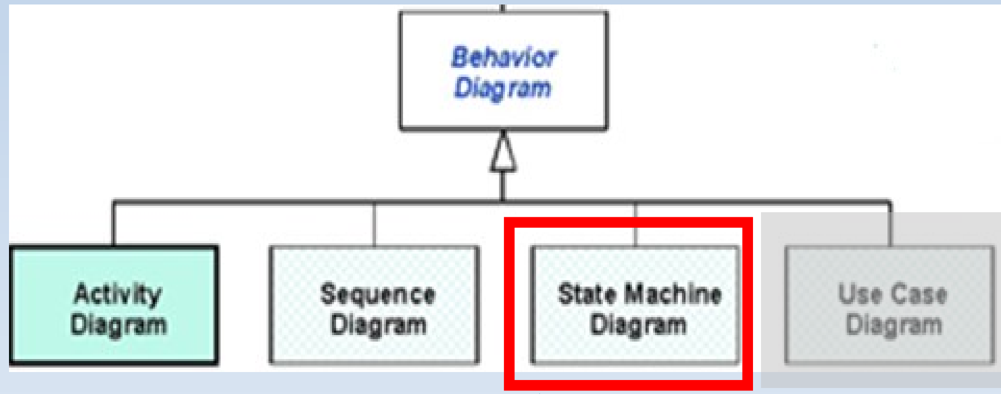
- **Event Based Behavior => State Machines**
- **Black-blox view => Use Cases**
- …also
  - Flow-Based Behaviour
  - Message Based Behaviour

Behaviour

UML



SysML

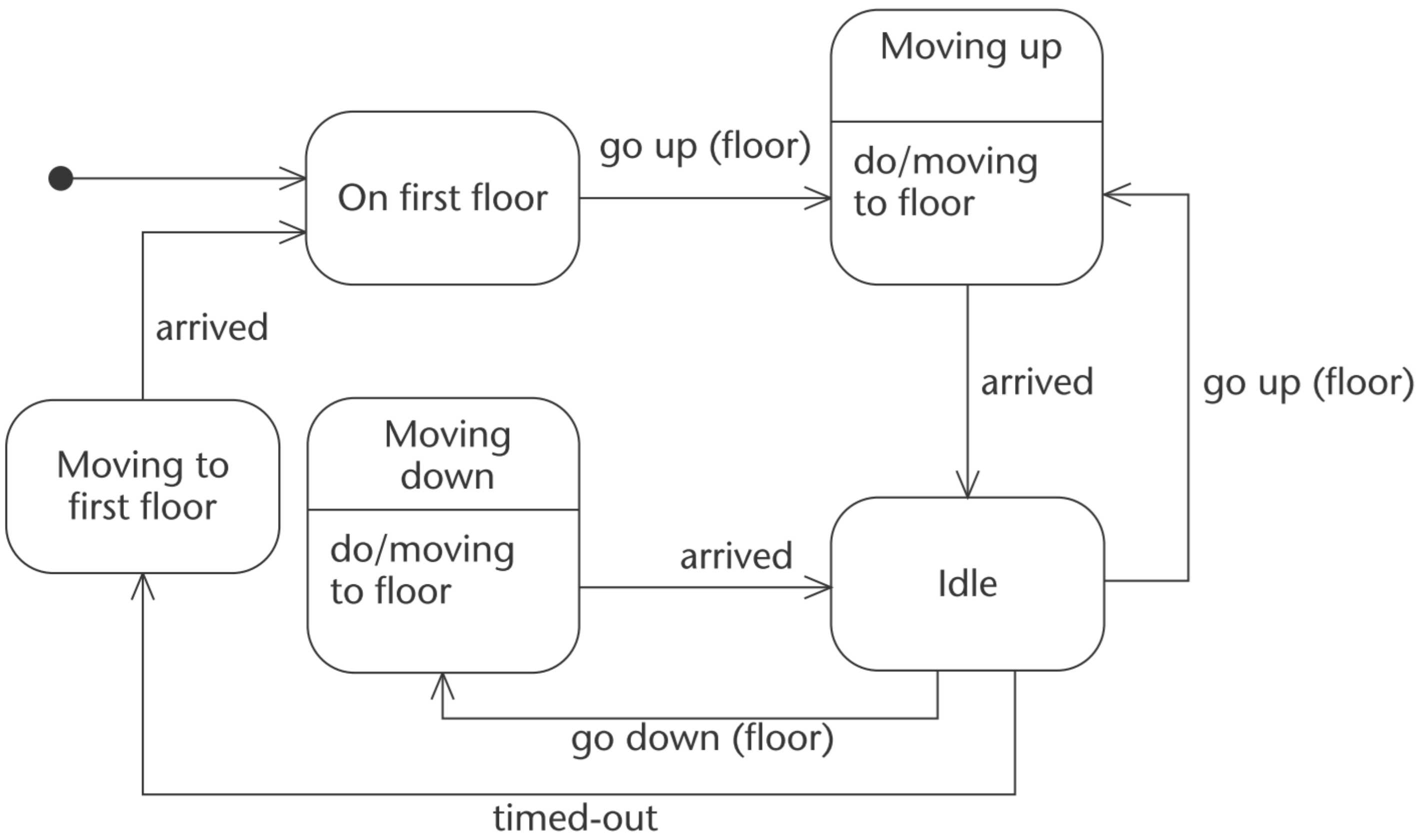


State Machines (Event-based behavior)

BTW, why not some fun ;-)
https://www.youtube.com/watch?v=ABA3TGQVhTg

State Machine Diagrams (UML & SysML)

- A state machine diagram models the lifecycle of an entity.
- It specifies how state changes as a response to events.



States

- A **state** represents a situation where an **invariant condition** holds.
- **Invariant conditions** are the properties that the system satisfies in every reachable state.

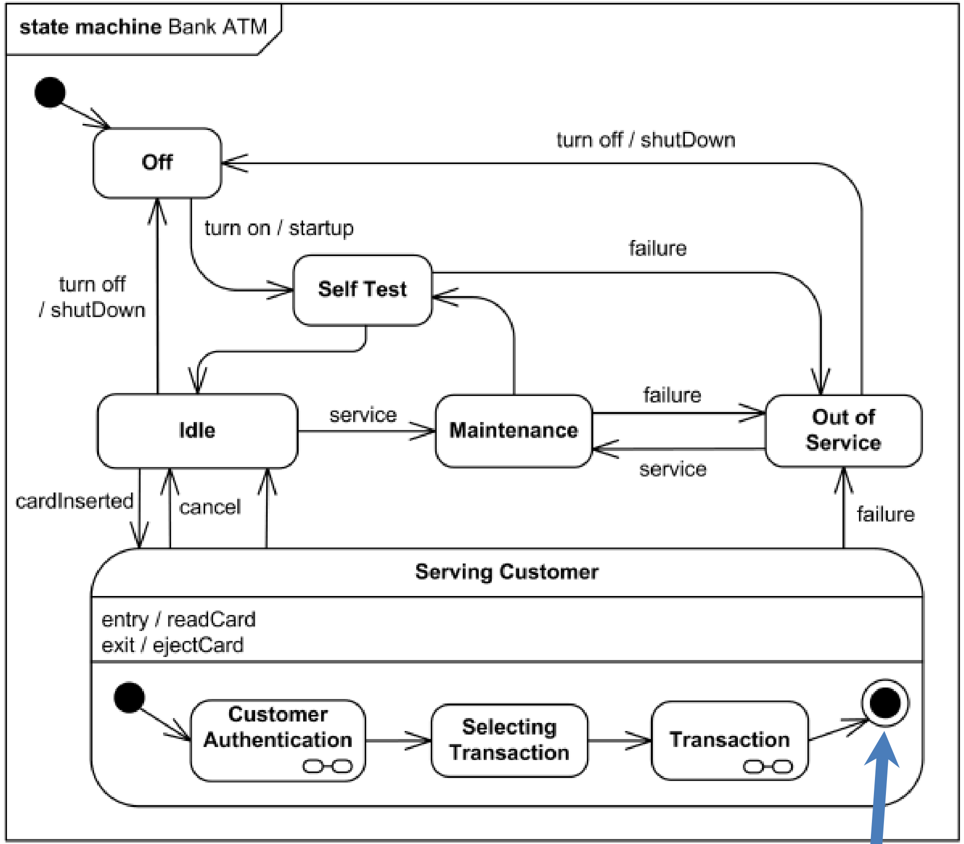
- An invariant condition can be:
  - **static** (e.g. waiting for an event)

  or
  - **dynamic** (e.g. performing a set of activities)

Entity Life Cycle

Transition diagrams define two kinds of constraints on entity life cycles:

- The set of allowed states. This is a static constraint.
- The set of legal sequences of these states. This is a transition constraint because it involves two or more states.



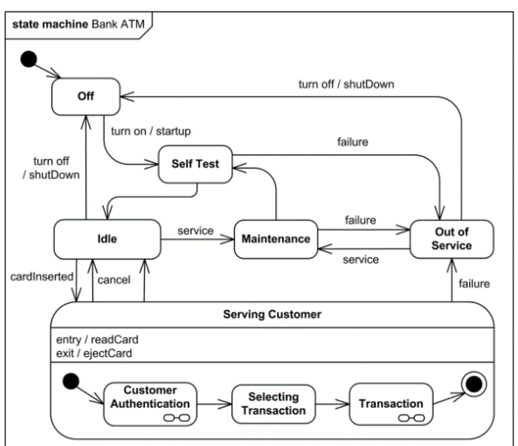
The **life cycle** of an entity **e** at time **t** is the sequence of states activated by **e** since its creation until **t**.

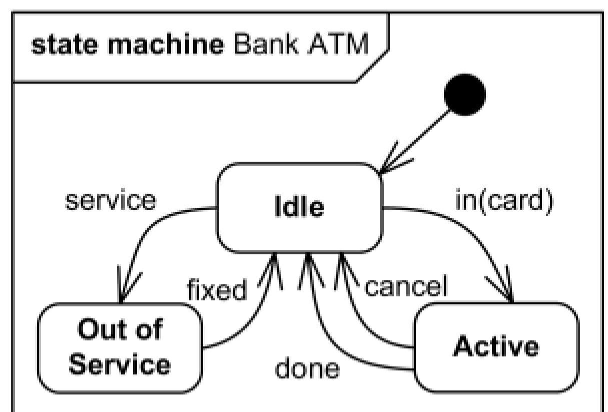
A life cycle is said to be **complete** if its last state is always a final state.

UML/SysML state machines

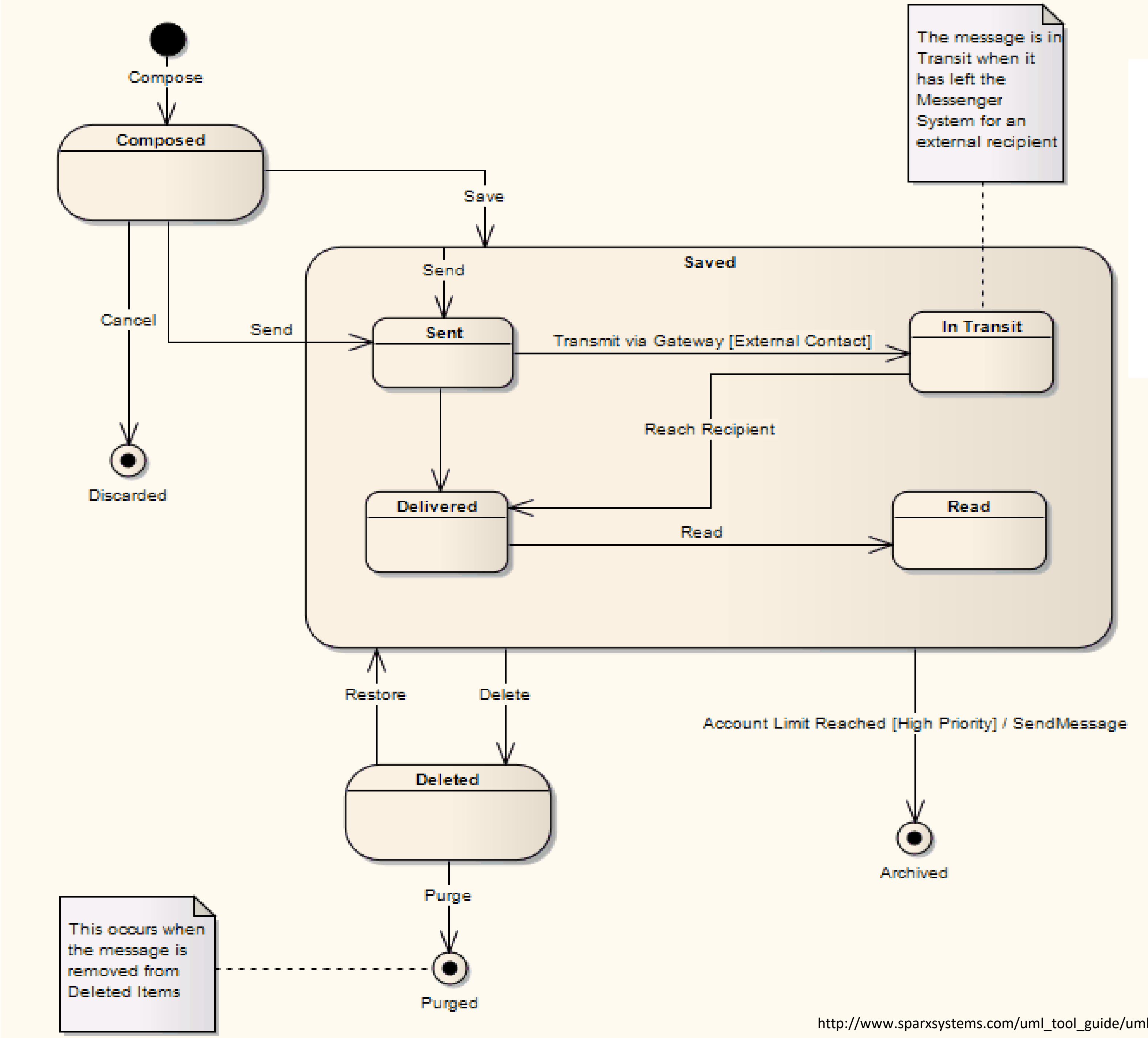
- Entity Behavior can be modeled as **state machines**.
- In principle, not all entities need to be modeled as state machines.
- Transition diagrams are an effective mechanism for defining the behavior of the instances of some entity types.
- However, they are not intended to be the best mechanism in all cases. This is one reason why some conceptual modeling languages offer several ways to model behavior.

Conceptual Modeling of Information Systems (chapter 13, page 301)





An example of a State Machine Diagram



http://www.sparxsystems.com/uml_tool_guide/uml_dictionary/statediagram.html

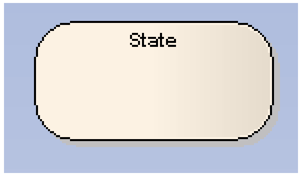
States...

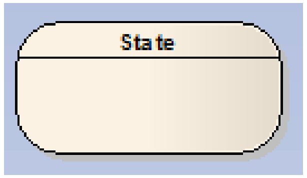
- The **State Machine** concept can be used for modeling discrete behavior through finite state transition systems.
- The state machine represents behavior as the state history of an object in terms of its transitions and states.
- The activities that are invoked during the transition, entry, and exit of the states, are specified along with the associated event and guard conditions.
- A composite state has nested states that can be sequential or concurrent

Simples States and Composite States

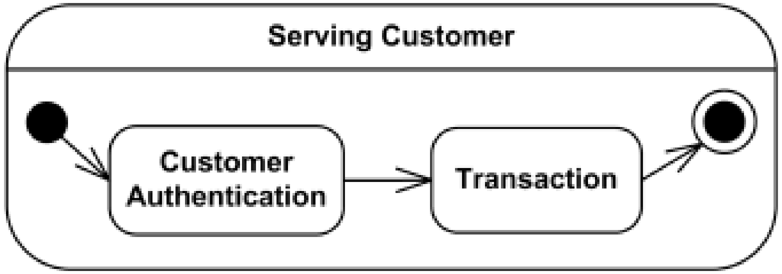
There are two types of states

– Simple States



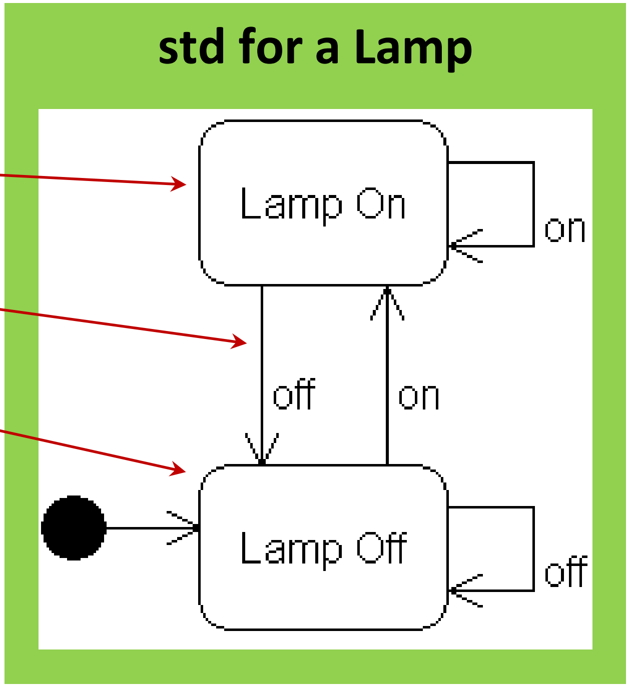


– Composite States: a state that contains other states (sub-states), allowing the construction of **hierarchical (or composite) state machines**



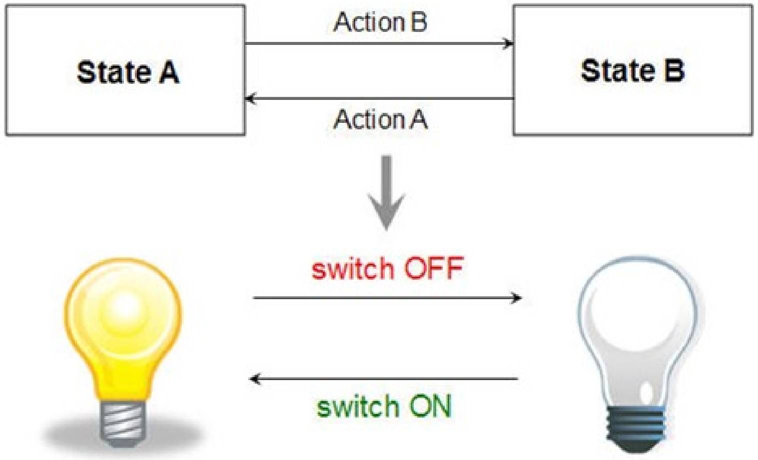
State Transitions

- A **transition** is based on:
  - a source state ***s***,
  - an event ***d***,
  - a target state ***t***.



- When the state machine is in state ***s*** and it receives event ***d*** for which there is a transition to state ***t***, then the transition is said to be *enabled.*
  - The event ***d*** is called the **trigger** of the transition.
  - One says that ***d*** **fires** the transition.

- If the machine receives an event which is not the trigger of any transition then the machine is unaffected by the event.

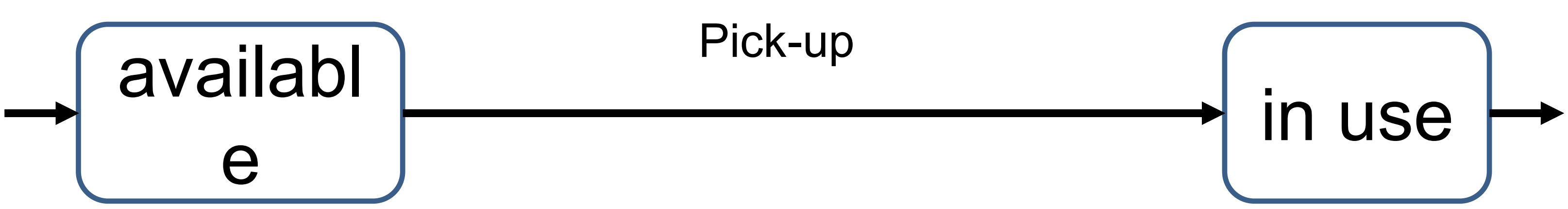


State Transitions

- An example of a transition of a rental car is:
  - **Source state**: *available*
  - **Trigger**: domain event *Pick-Up*
  - **Target state**: *in use*

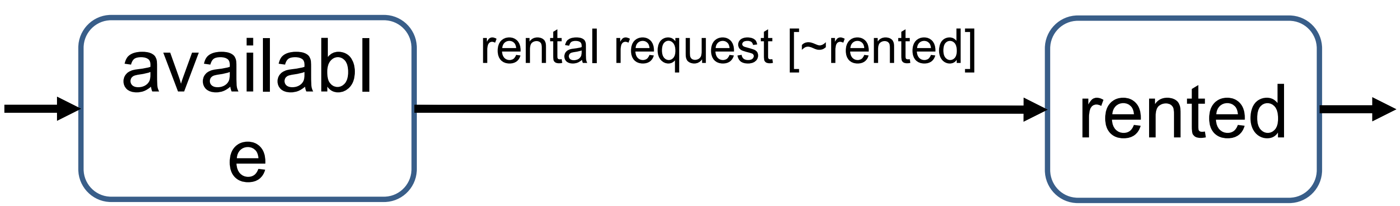
The meaning is that:

- When a car is in the state *available* …
  - and the domain event *Pick-Up* of that car occurs,
  - Then, the new state of the car is *in use*.



State Transitions and Guards

- Example of a transition of a rent car with a guard is:
  - **Source state**: *available*
  - **Guard**: the car is not booked for maintenance
  - **Trigger**: domain event *Rental Request*
  - **Target state**: *rented*
- The meaning is that:
  - when a car is in the state *Available*
  - and the domain event *Rental Request* occurs,
  - if the car is not booked for maintenance
  - then the new state of the car is *rented*.



transition ::= [*triggers*] [*guard*] ['/' *behavior-expression*]
triggers ::= *trigger* [',' *trigger*]*
guard ::= '[' *constraint* ']'

State Transitions (UML / SysML)

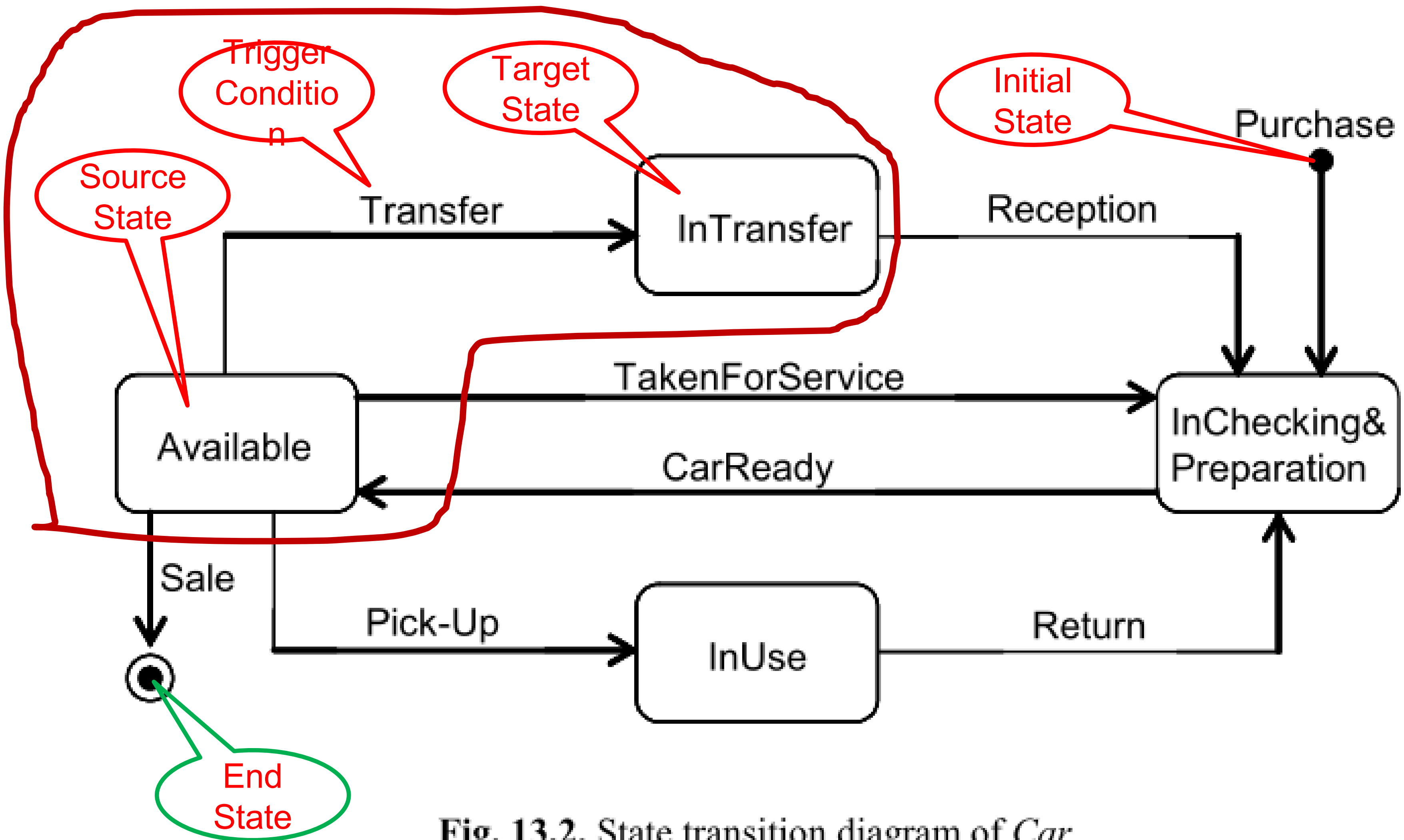


Fig. 13.2. State transition diagram of *Car*

Conceptual Modeling of Information Systems (chapter 13, page 306)

Example: Telephone (partial)

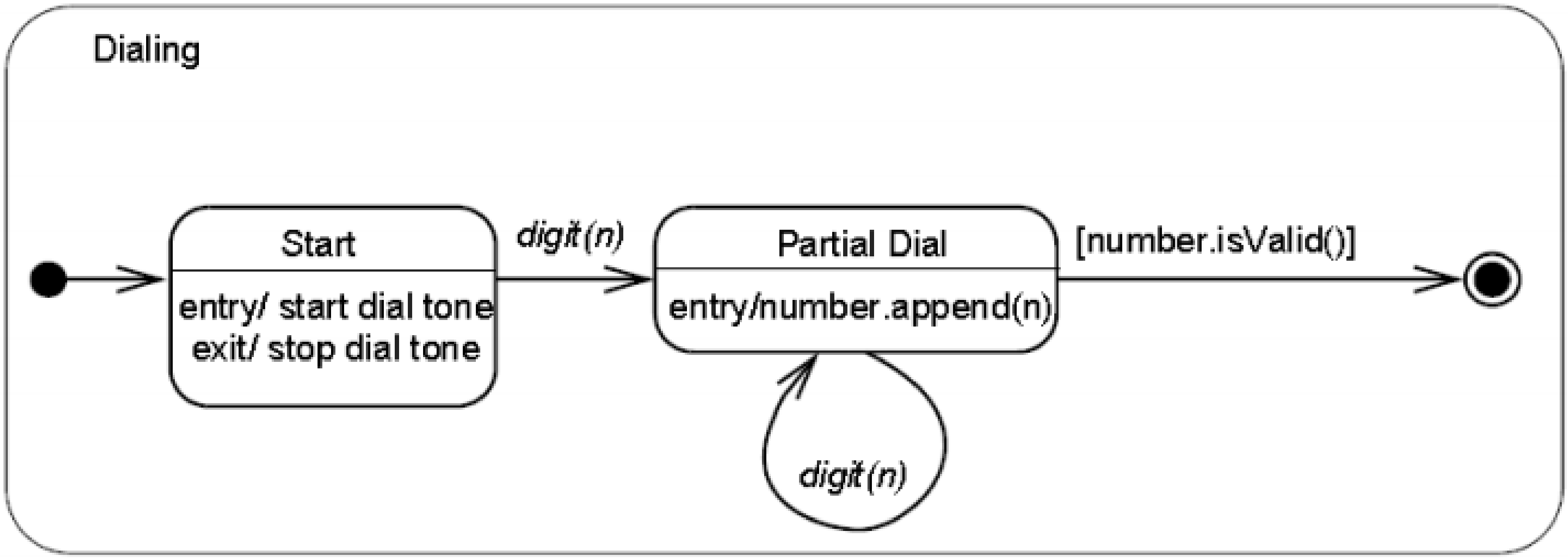


Figure 14.7 Composite State with two States

Example: Telephone

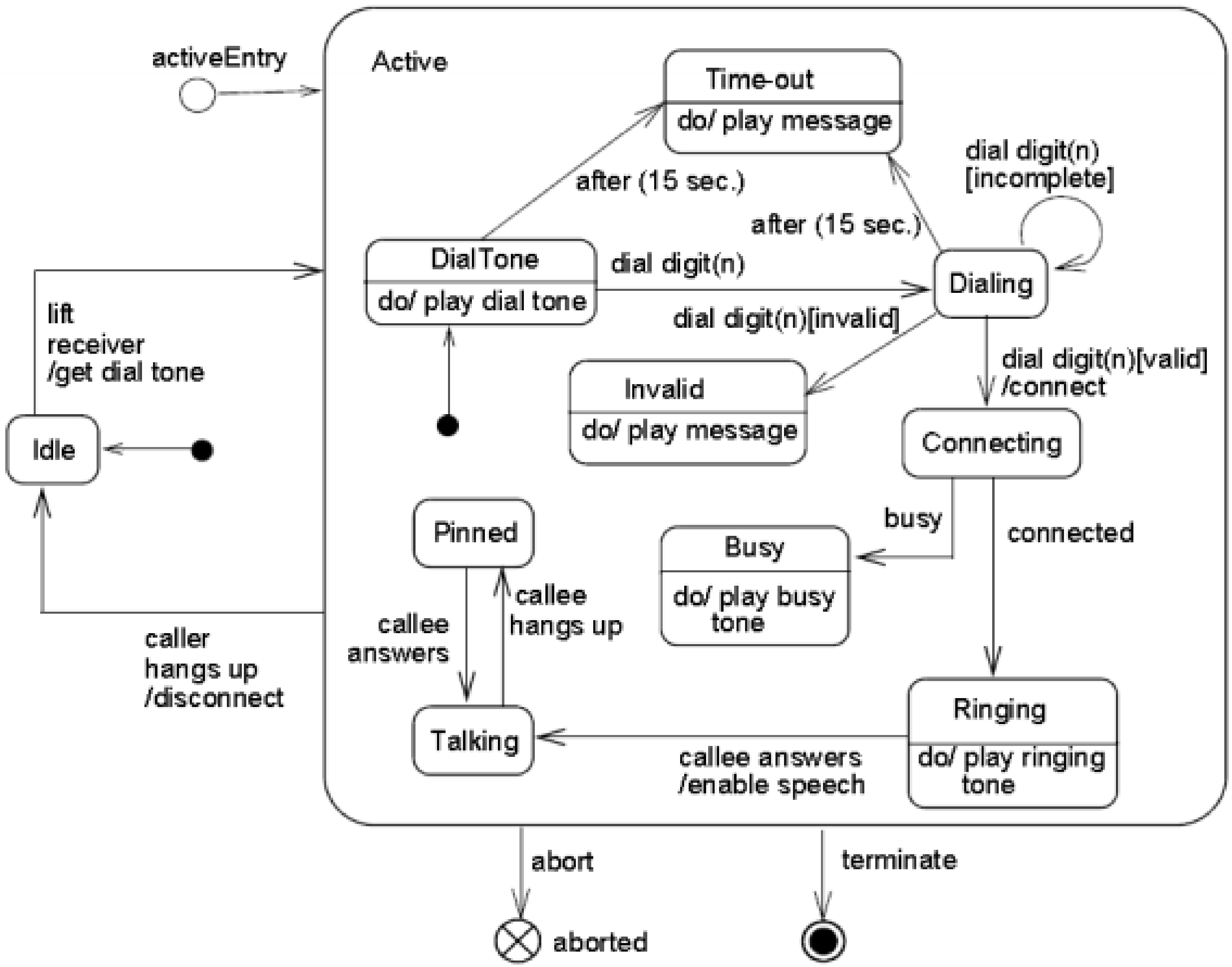
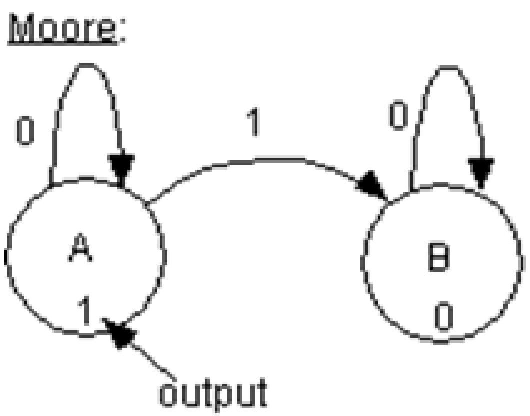
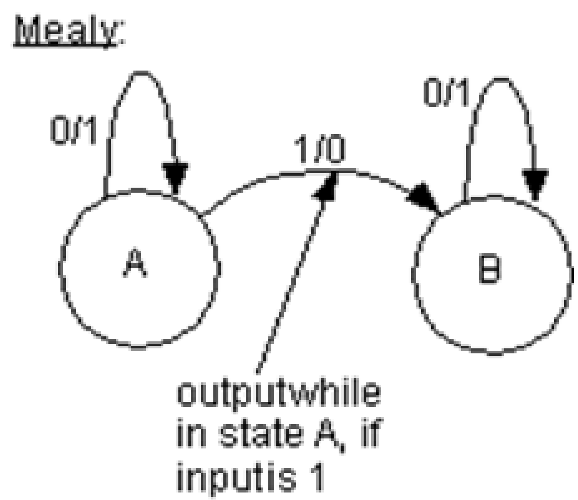


Figure 14.36 StateMachine diagram representing a telephone

Remembering Moore and Mealy Machines

A **Moore machine**	A **Mealy machine**
associates an output to a state.	**associates an output to a transition**.
Each state has exactly one output. Every time the machine reaches a target state it produces the output associated with that state.	Every time the machine makes a transition to a target state it produces the output associated with that transition.





It can be shown that for each Moore machine there exists at least one Mealy machine that produces the same outputs for all possible inputs, an vice-versa. Hence, Moore and Mealy machines are said to be equivalent.

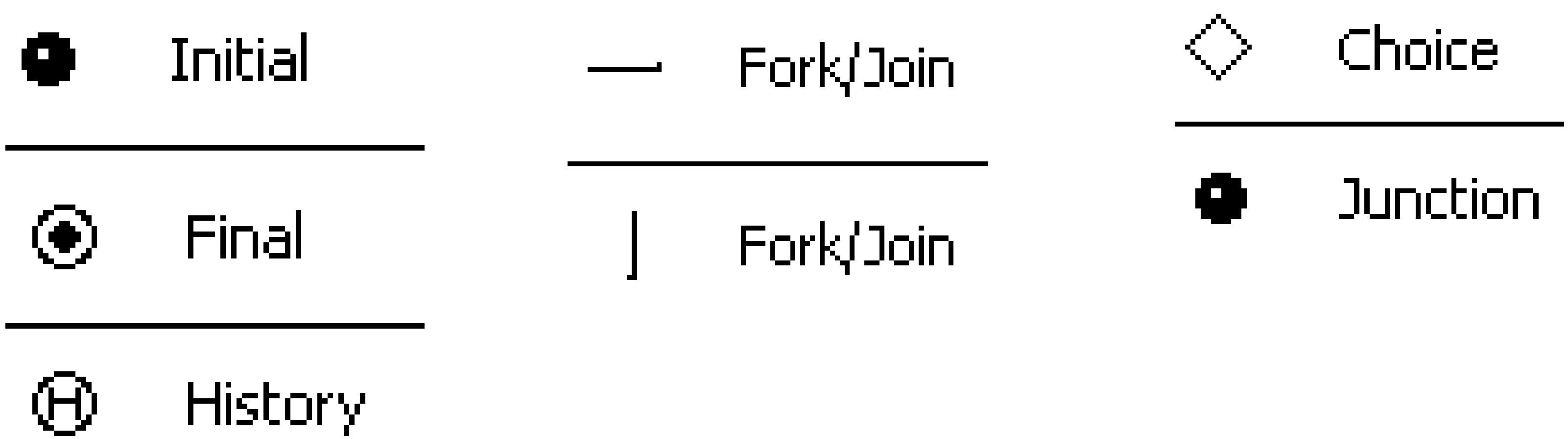
UML/SysML state machines

- UML/SysML state machines have the characteristics of both Mealy machines and Moore machines.

- They support entry and exit actions, which are associated with states rather than transitions, as in Moore machines.
- They support actions that depend on both the state of the system and the triggering event, as in Mealy machines.

Pseudostates

- A **pseudostate** is an abstract state used to represent the start, end or internally to connect other stated of the diagram.
- The following pseudostates are defined in UML/SysML.



Pseudostates

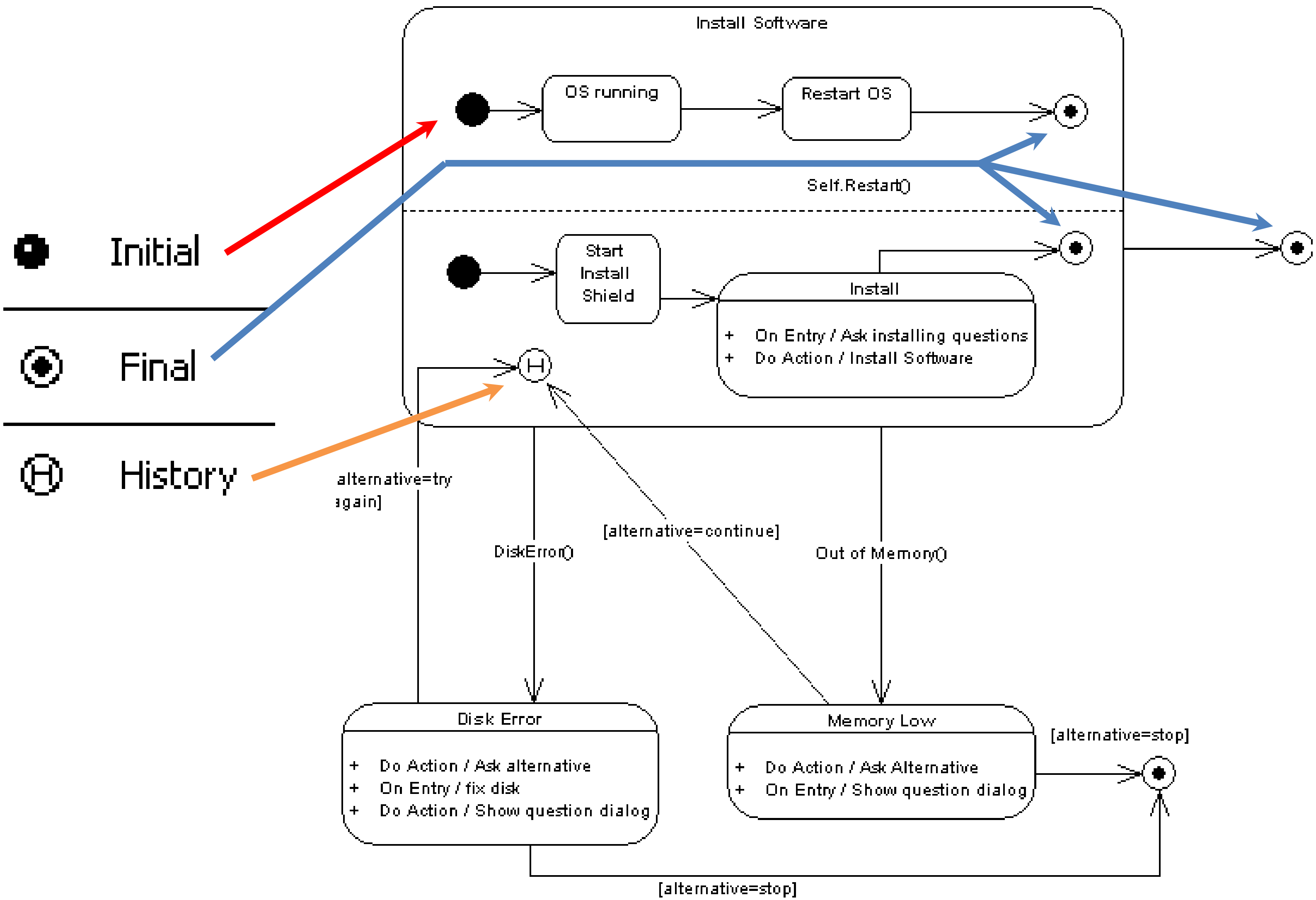
- **Initial state** – Represents the first state that occurs when entering a region. An initial state must be present in each region.

- **Terminal/Final state –** Represents the final state of a region and ends the execution of all states within that region.

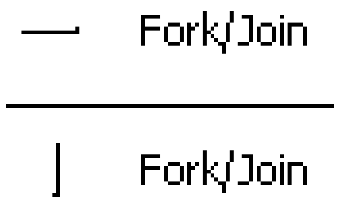
- **History –** Specifies that, when the statechart leaves and then returns to a region, the statechart enters the substate that was active when the statechart left the region.

  Cf. **shallow** and **deep** history on the book.

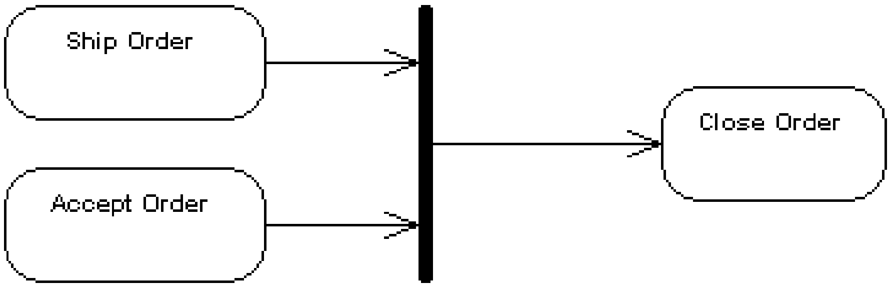
Pseudostates



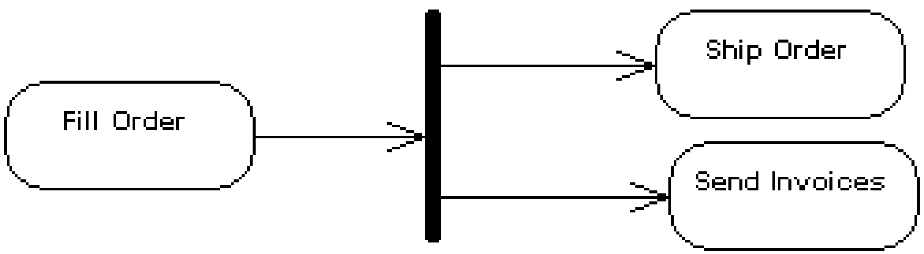
Pseudostates



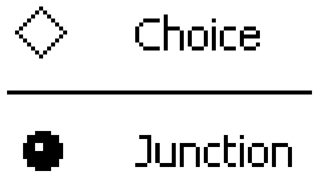
- **Join** - serve to merge several transitions, with no guards, emanating from one or more source states.

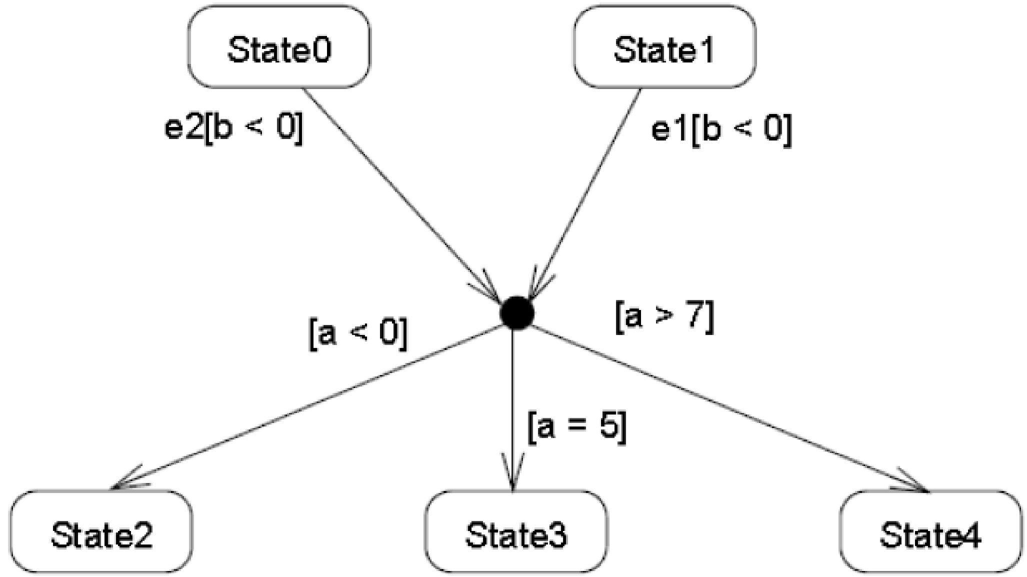


- **Fork** - serve to split an incoming transition, with no guard, into two or more transitions.



Pseudostates - Junction

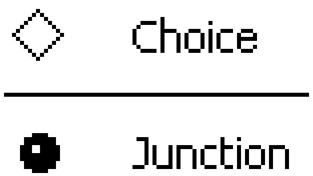


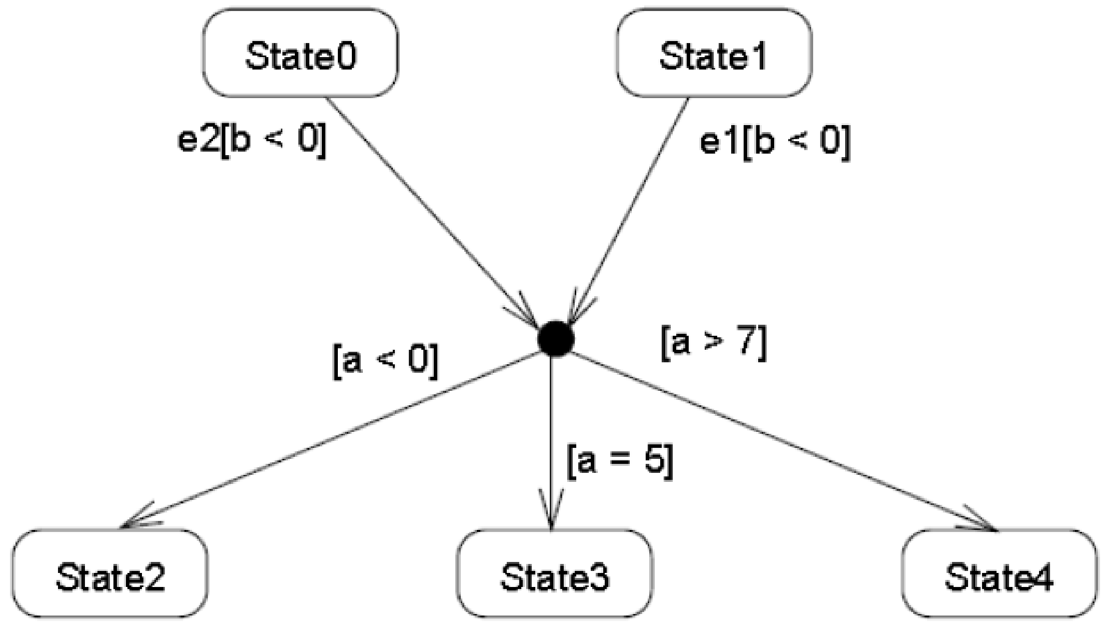


Junction

- A pseudostate with at least one incoming and one outgoing transition. In general, the incoming transitions have a trigger, and both the incoming and the outgoing transitions have a guard.
- A junction provides a means to simplify two or more transitions by factoring out their common parts. For example, a junction can be used to converge multiple incoming transitions into a single outgoing transition representing a shared transition path.
- Conversely, a junction can be used to split an incoming transition into multiple outgoing transition segments with different guard conditions.

Pseudostates - Junction

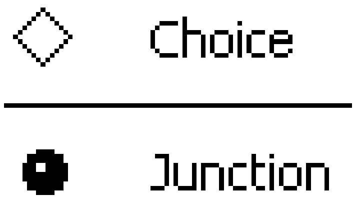




Example of a junction to simplify four transitions:

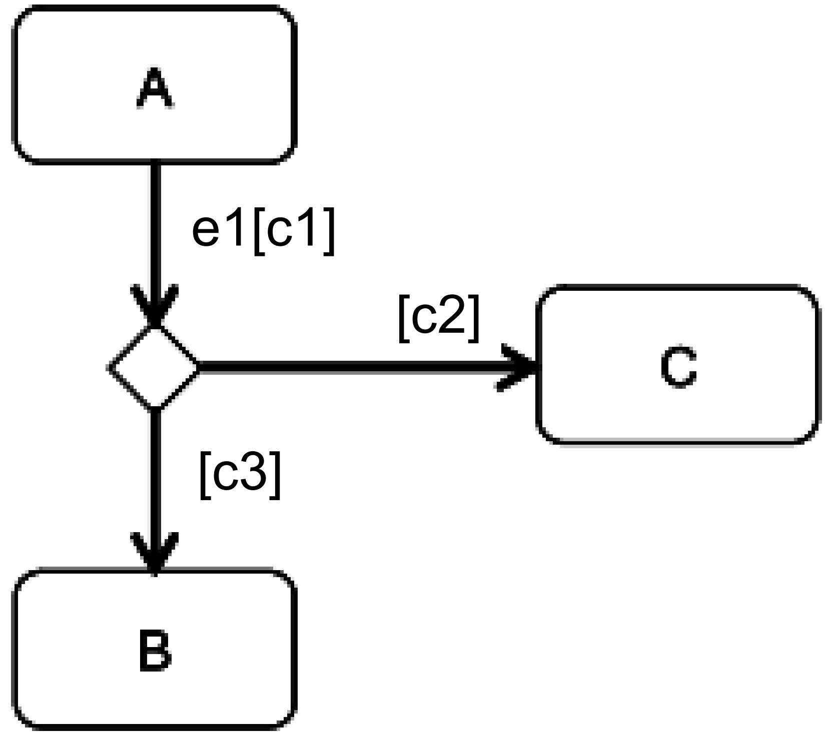
- The transition from State0 to State3 triggered by e2 and guarded by the condition [“b<0” and “a=5”]
- The transition from State0 to State4 triggered by e2 and guarded by the condition [“b<0” and “a>7”]
- The transition from State1 to State2 triggered by e1 and guarded by the condition [“b<0” and “a<0”]
- …

Pseudostates - Choice

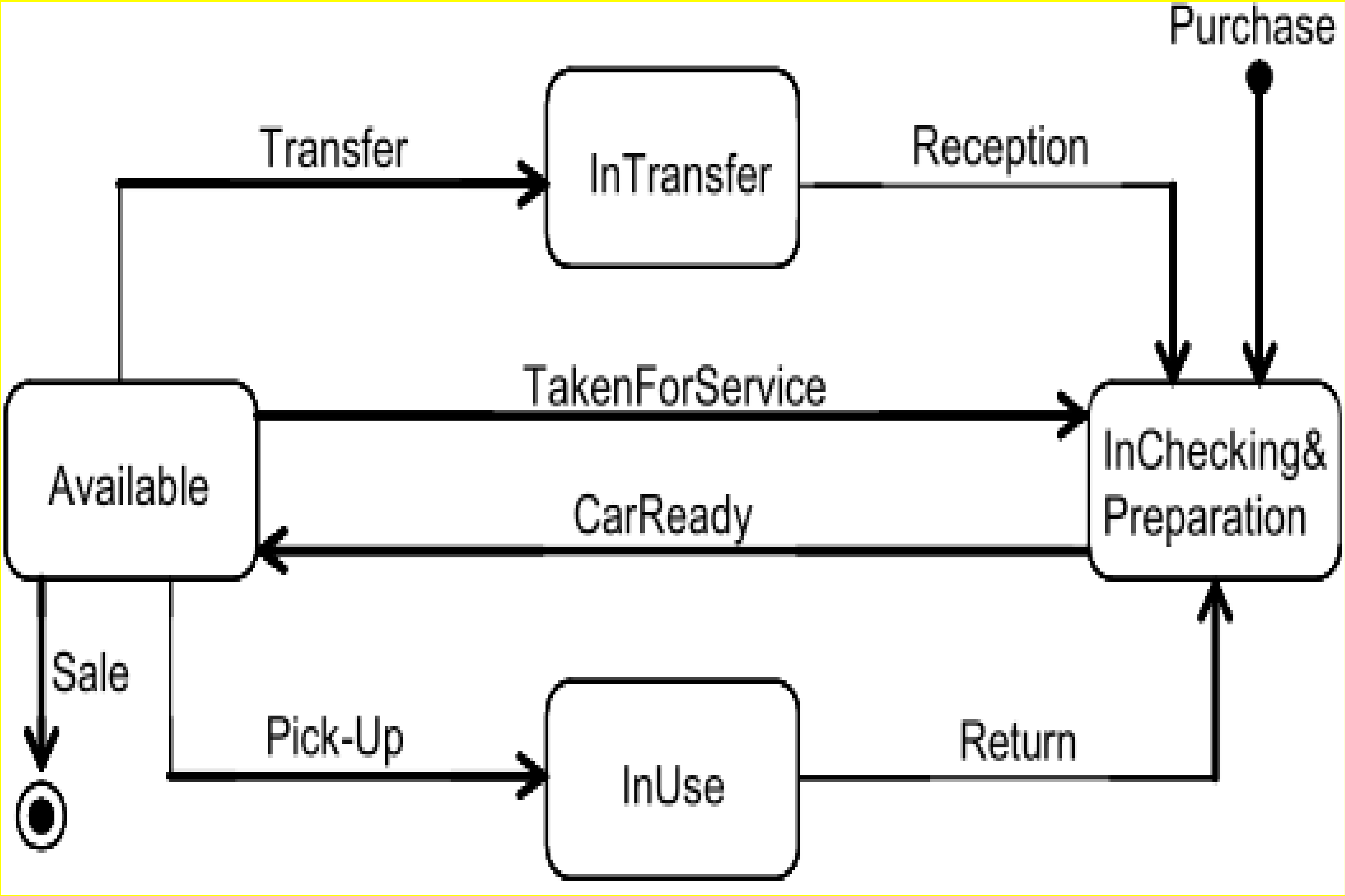


Choice

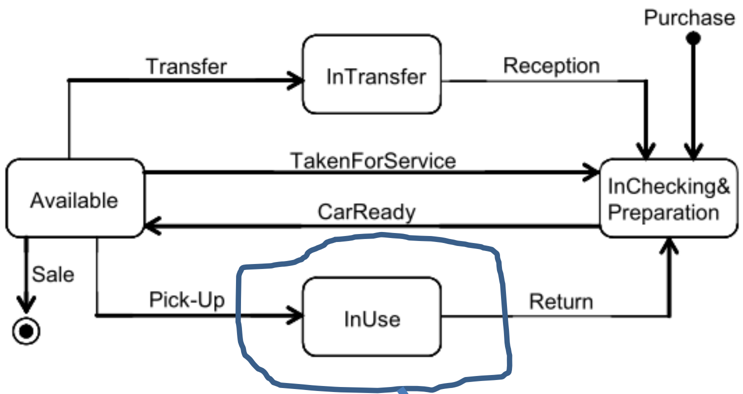
- Similar to a junction, but the guards of the outgoing transitions are evaluated once the incoming transitions have produced their effect.
- Choices are considered as dynamic conditional branches because the target state is not known until the operations associated with the incoming transitions have been completed.



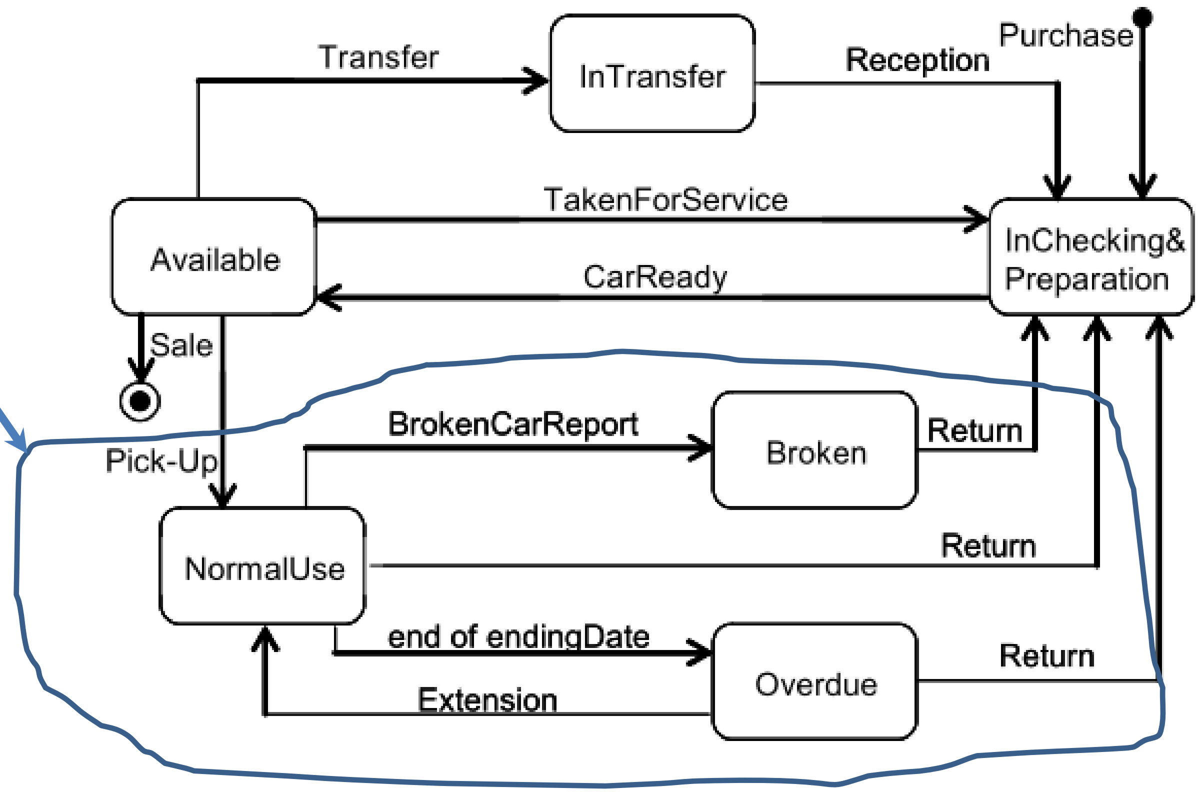
Example: Car rental



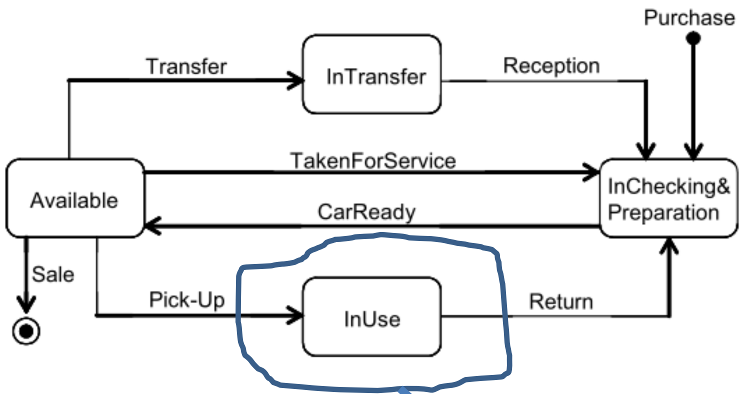
Detailing states

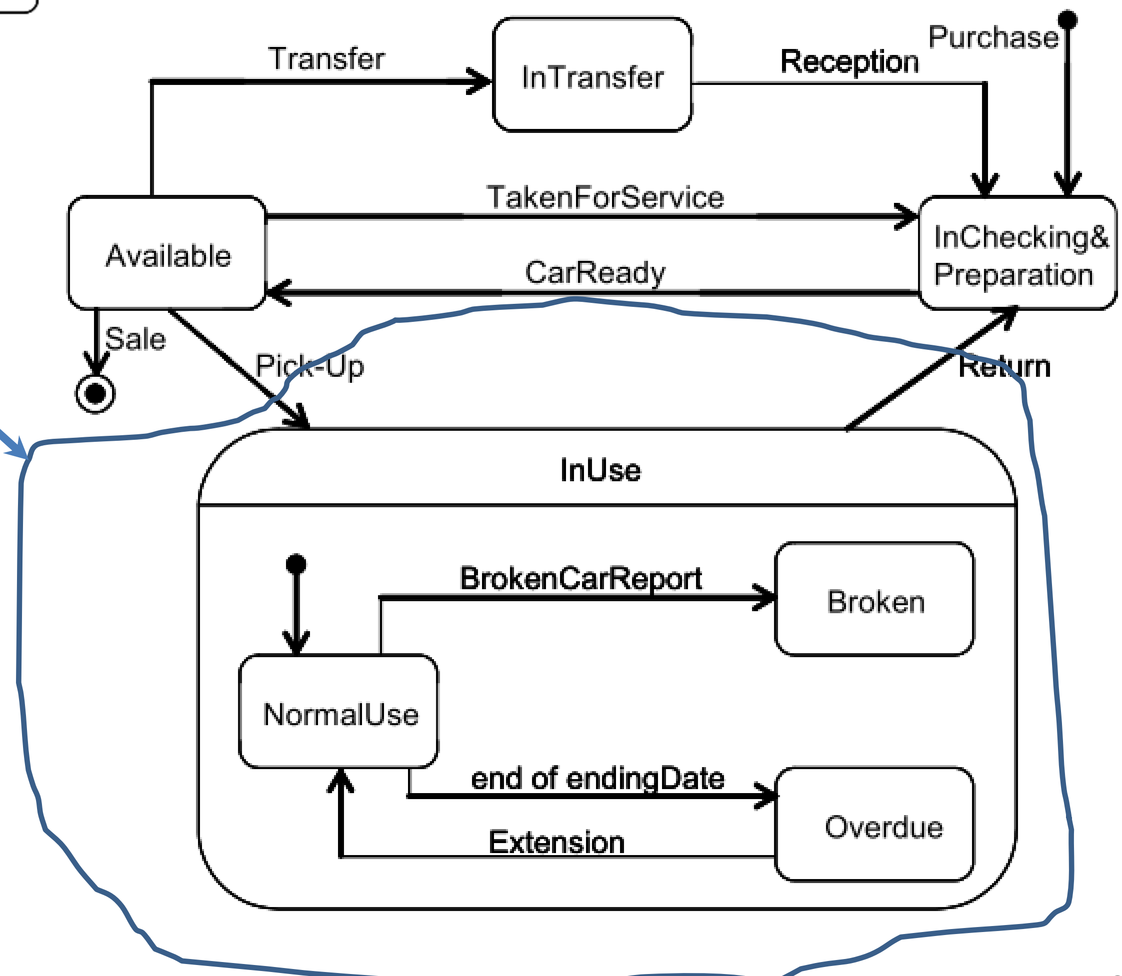


The problem of this approach: it forces to change the high level conceptualization (notion of “InUse” state is lost, reducing semantics…)…



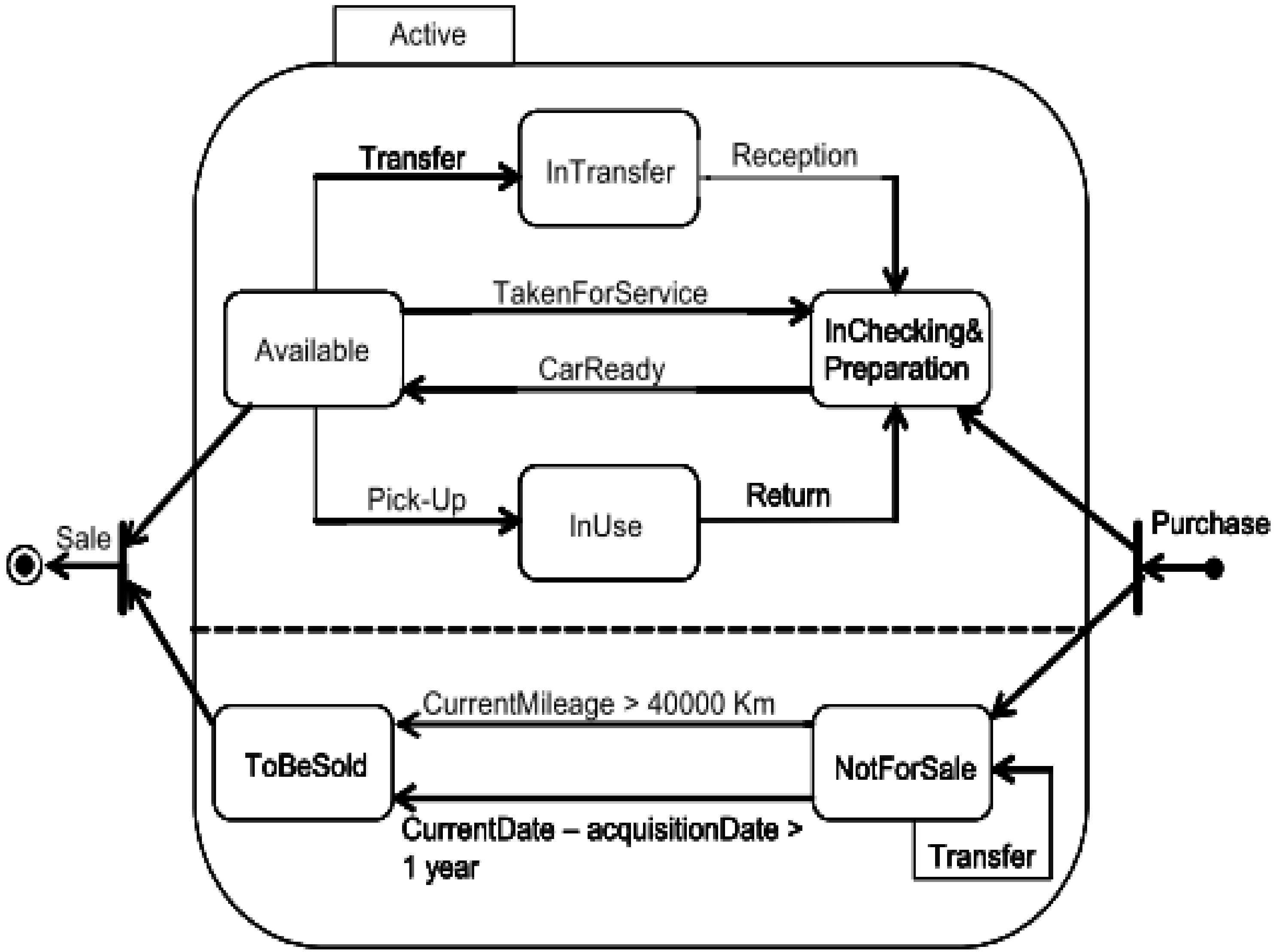
Detailing states





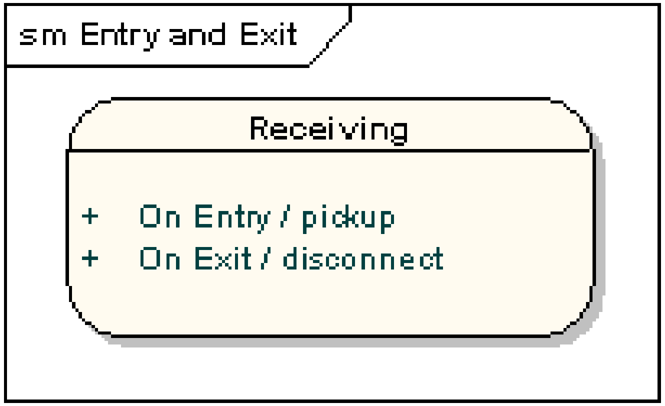
With a Composite State, “InUse” state is kept, stressing its semantics!!!

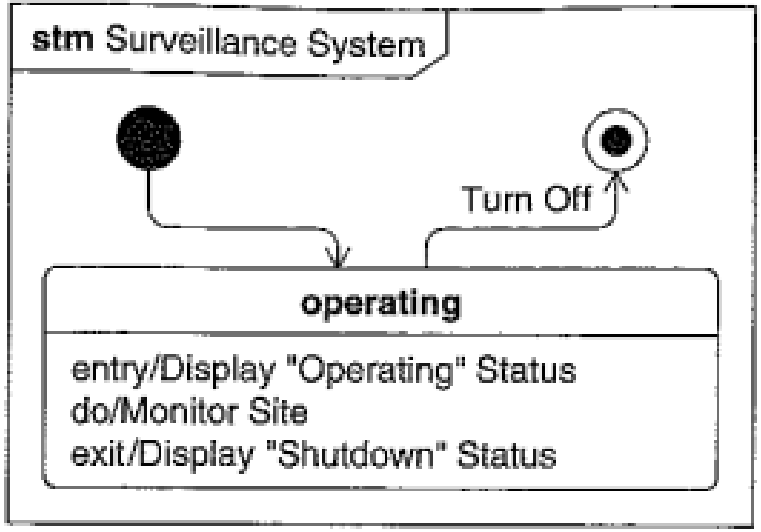
Parallelism



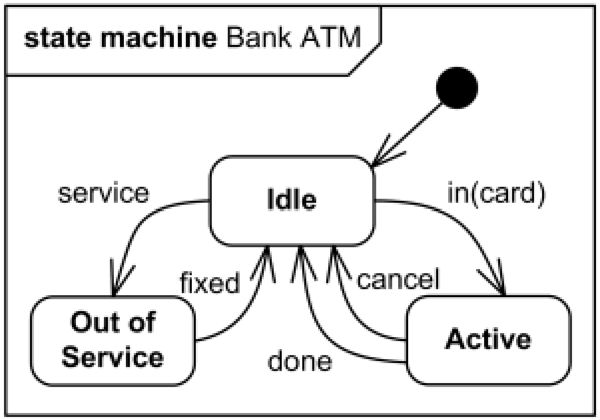
States and Behaviours

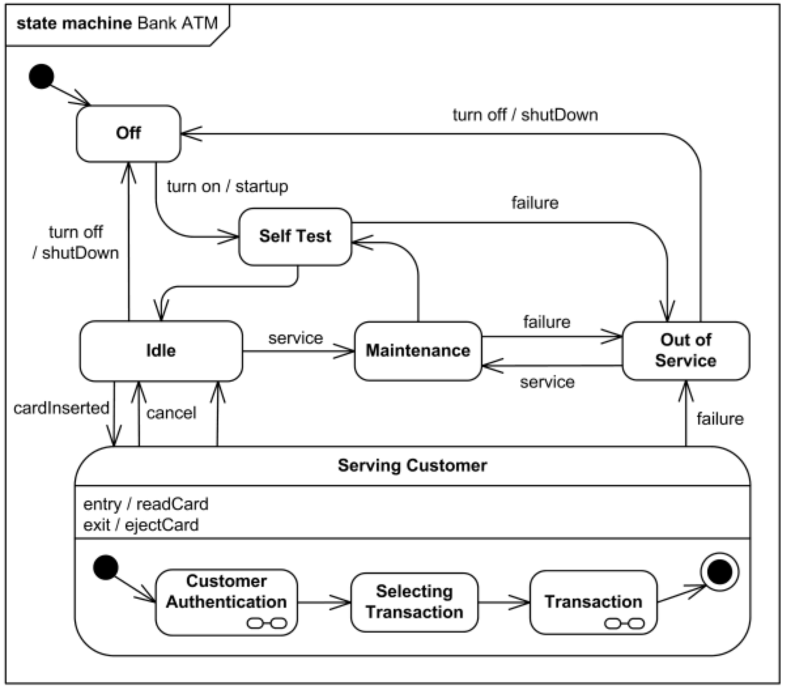
- Behaviours
  - **entry**/…action…
  - **do**/…action…
  - **exit**/…action….

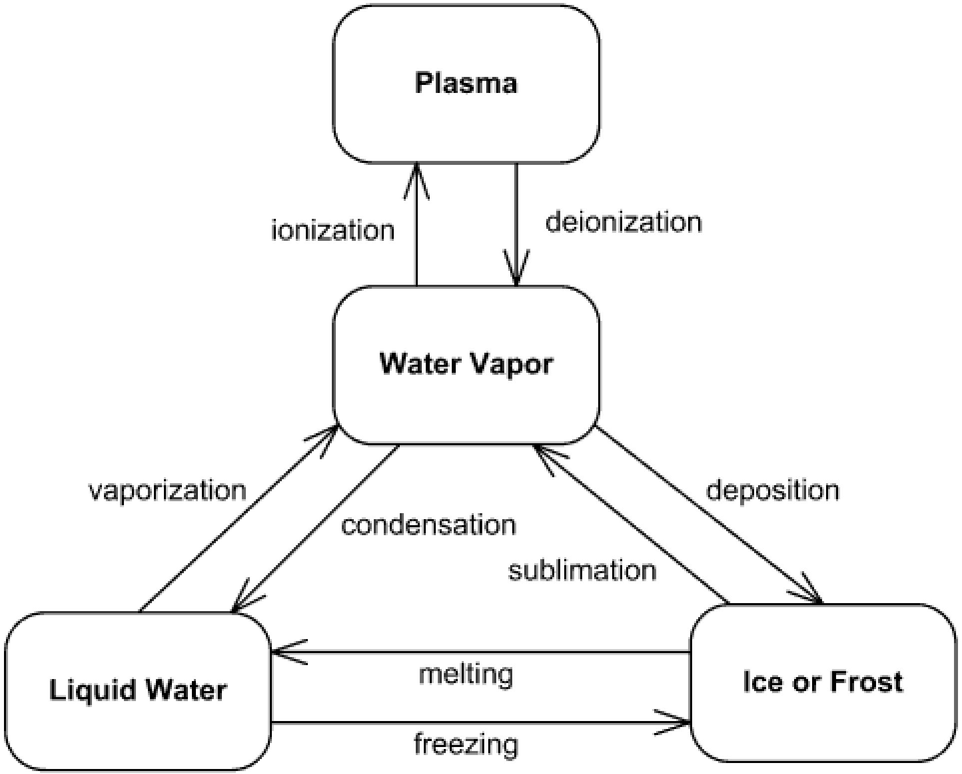




State Diagrams as Event-based Behaviour

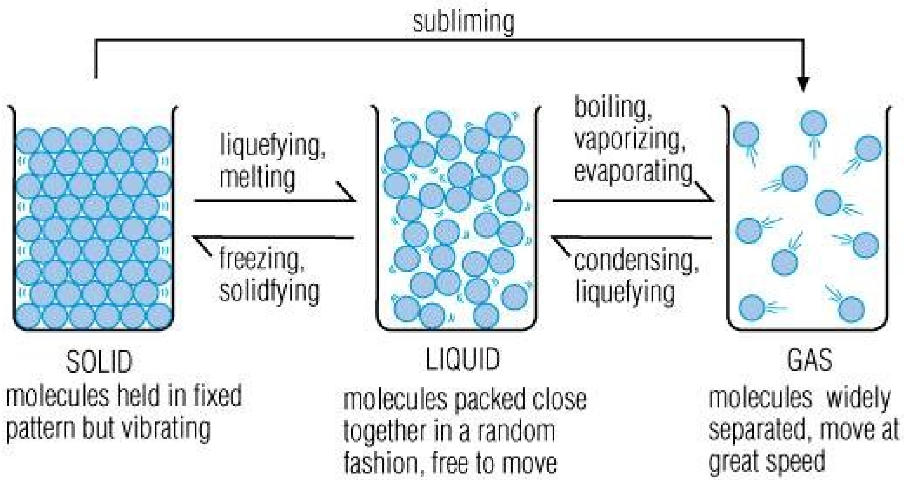


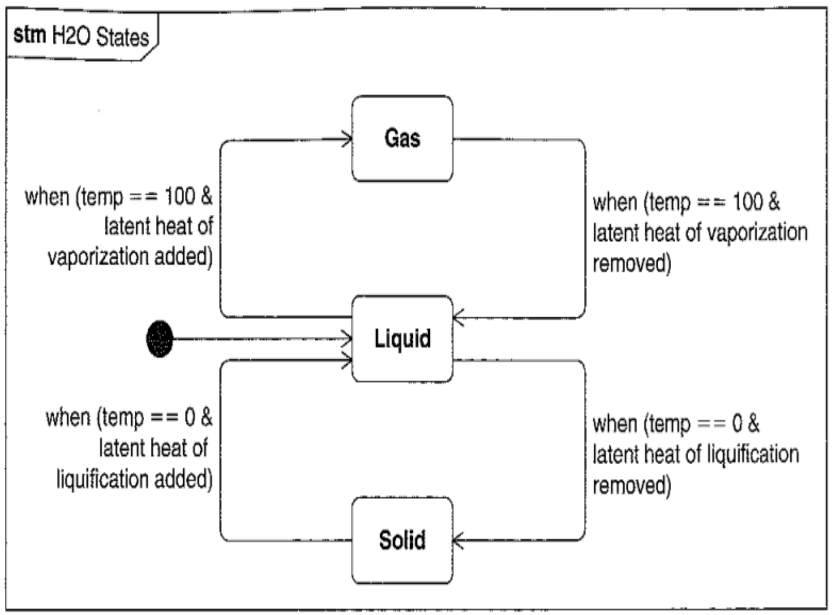


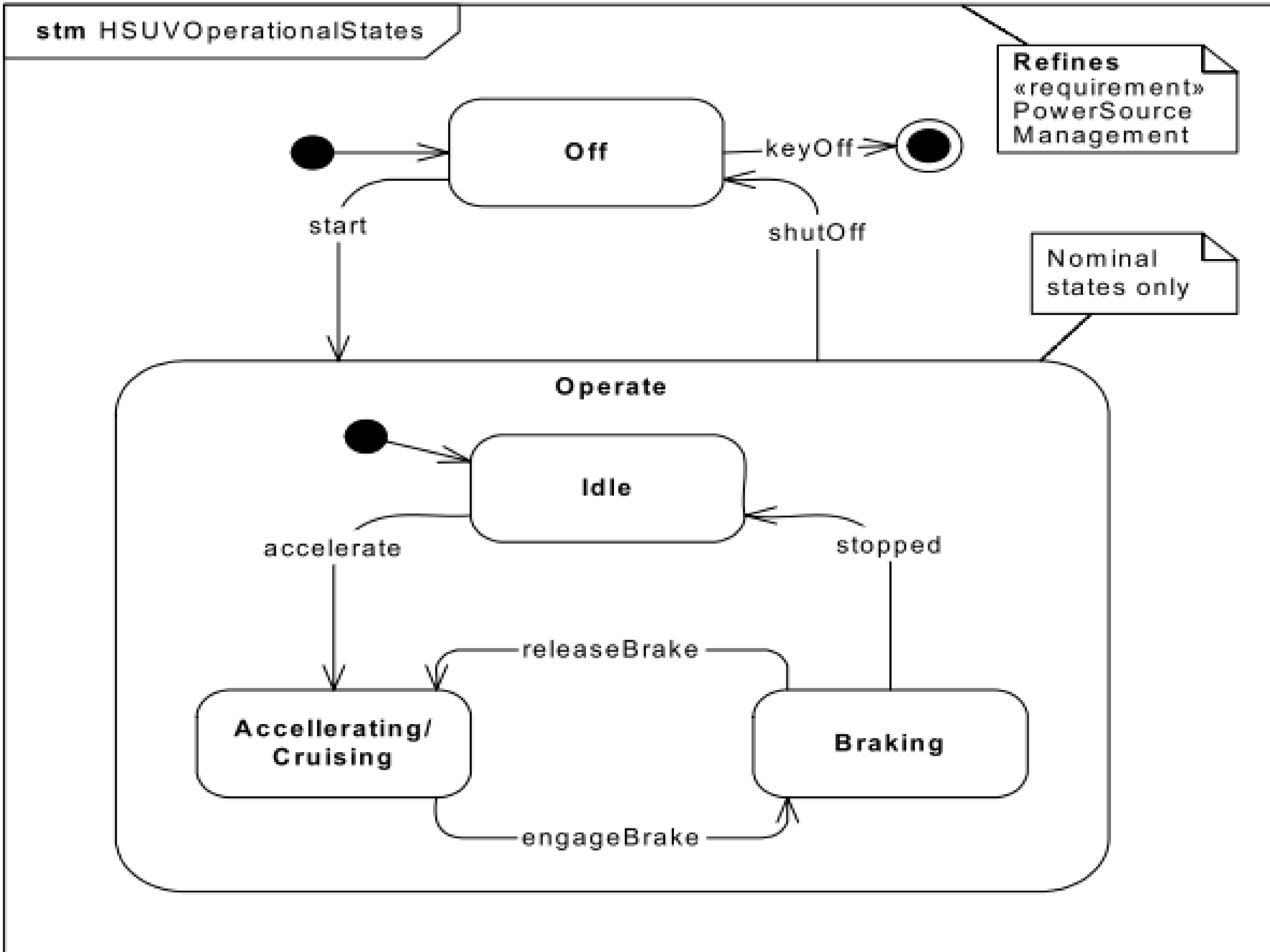


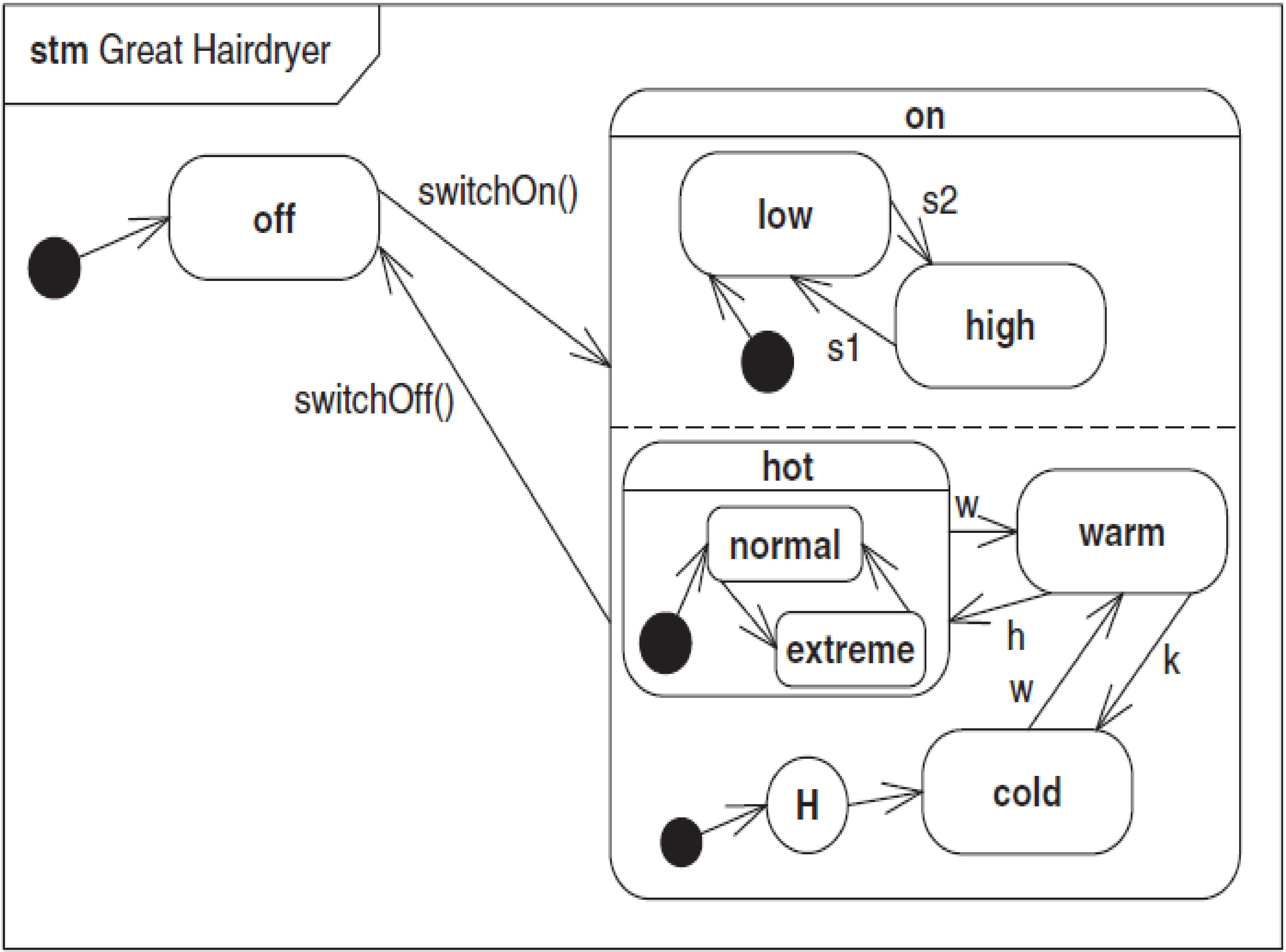
H_2O

http://media.tiscali.co.uk/images/feeds/hutchinson/ency/c00503.jpg

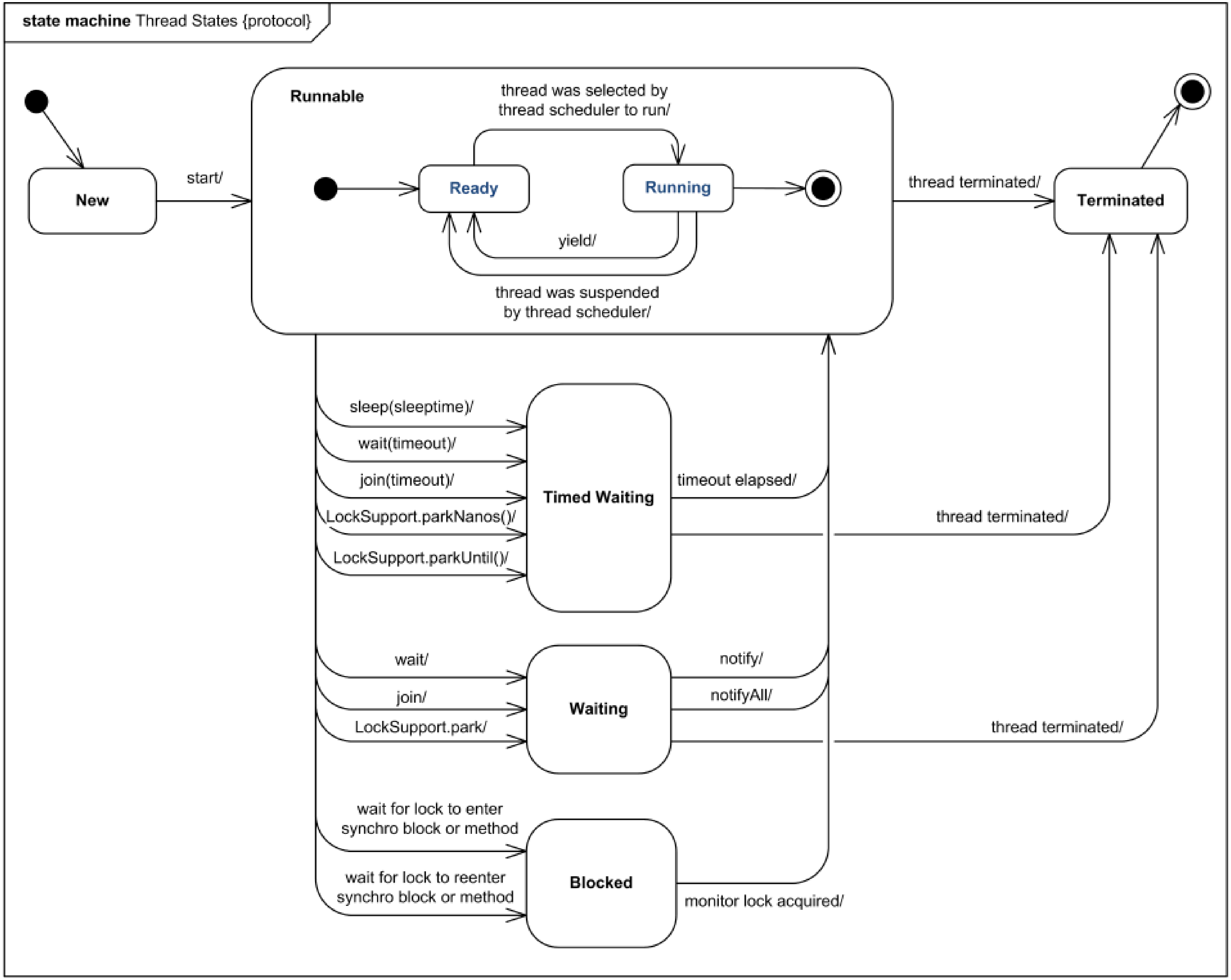





stm HSUVOperationalStates
Refines
«requirement»
PowerSource
Management
Off
keyOff
start
shutOff
Nominal
states only
Operate
Idle
accelerate
stopped
releaseBrake
Accellerating/
Cruising
Braking
engageBrake



stm Great Hairdryer
off
switchOn()
switchOff()
on
low
s2
high
s1
hot
normal
extreme
w
warm
h
w
k
H
cold


state machine Thread States {protocol}
Runnable
thread was selected by
thread scheduler to run/
New
start/
Ready
Running
yield/
thread was suspended
by thread scheduler/
thread terminated/
Terminated
sleep(sleeptime)/
wait(timeout)/
join(timeout)/
LockSupport.parkNanos()/
LockSupport.parkUntil()/
Timed Waiting
timeout elapsed/
thread terminated/
wait/
join/
LockSupport.park/
Waiting
notify/
notifyAll/
thread terminated/
wait for lock to enter
synchro block or method
wait for lock to reenter
synchro block or method
Blocked
monitor lock acquired/

State Machine Diagrama Overview