Object Oriented Analysis & Design

Week 8

Statechart Diagrams

Lecturer: Dr. Msagha J Mbogholi, PhD

Flashback from Lesson 7

- The structural model is described using two other models: the static model and the dynamic model.
- Static analysis is concerned with what the system should do. In doing so we differentiate this with "how" the system should do it. This is done through the static model which is a design implementation of static analysis.
- Object identification is done using the following techniques: textual analysis, finding "uses", finding players, finding generalization, finding analogies, finding reuse, brainstorming, and using RUP stereotypes.
- The UML modeling elements in class diagrams are: Classes, their structure and behavior; relationships components among the classes like association, aggregation, composition, dependency and inheritance; multiplicity and navigation indicators; role names or labels.
- A constraint restricts the values that entities can assume.
- Object diagrams are used to: visualize, specify, construct, and document, the existence of certain instances in a system, together with their relationships.

Content

- Introduction
- Events, states and transitions
- Complex states, entry and exit actions
- Advanced state machines

Part 1

Introduction

Introduction

- The statechart diagram is not a diagram that is used in every problem domain due to its uniqueness. It models the dynamic behavior of the system.
- This is because this diagram is only targeted at objects that undergo transition from one state to another; what does this mean?
- This is the essence of this lesson; to understand the different terms associated with statechart diagrams, how to use them, and when to apply them.
- In OO there are several problem domains where you will find there is no need for a statechart diagram; indeed there are more domains that don't use statechart diagrams than those that do.

Introduction (cont'd)

- Ojo and Estevez (2005) describe the statechart diagram as follows:
- "A statechart diagram defines the behavior of a single object or of a set of objects related by a collaboration.
- It captures the changes in an object throughout its lifecycle as they occur in response to internal and external events.
- The scope of a statechart is the entire life of one object."
- The UML includes statechart diagram notation to illustrate the events and states of things—transactions, use cases, people, and so forth. The use of statechart diagrams is emphasized for showing system events in use cases, but they may additionally be applied to any class. (Larman, 2002)

Introduction

- The UML specifies two types of statechart diagrams:
- Behavioral state machines – use state, events and transitions to specify behavior. These states can specify one or more actions that execute when the state is entered, resided in, or exited.
- Protocol state machines – use state, events and transitions to define the protocol of the context classifier (that is, a class, use case, sub system or entire system). The protocol will include:
  - Conditions under which the operations may be called on the classifier and its instances;
  - Results and ordering of operation calls.
- Protocol state machines do not include implementation details – they only describe how the behavior appears to an external entity. The states in protocol state machines can't define actions; this is the work of the behavioral state machines (the focus of this lesson).
- (Arlow and Neustadt, 2013)

Part 2

Events, states and transitions

2.1 Components

- Statechart diagrams are composed of the following components, which are discussed in detail in the remaining part of this section:
- States
  - Initial state
  - Final state
- Events
  - guard conditions
  - actions
  - event syntax
- Complex states
  - activities
  - entry actions
  - exit actions

2.2 State

- **A state** is the condition of an object at a moment in time—the time between events. An example of state is a telephone is in the state of being "idle" after the receiver is placed on the hook and until it is taken off the hook. (Larman, 2002).
- State is the current condition of an object reflected by the values of its attributes and its links to other objects. States can be initial or final (Ojo and Estevez, 2005)
- The notation for state is a rectangle with rounded corners, as shown in fig 1.

Fig 1. State (Ojo and Estevez, 2005)

2.2.1 Initial/start state

- The initial state identifies the state in which an object is created or constructed. Each state diagram must have one and only one start state.
- The initial state is called a pseudo-state because it does not really have the features of an actual state, but it helps clarify the purpose of another state of the diagram.
- The notation for the initial/start state is a filled solid circle, as shown in fig 2.

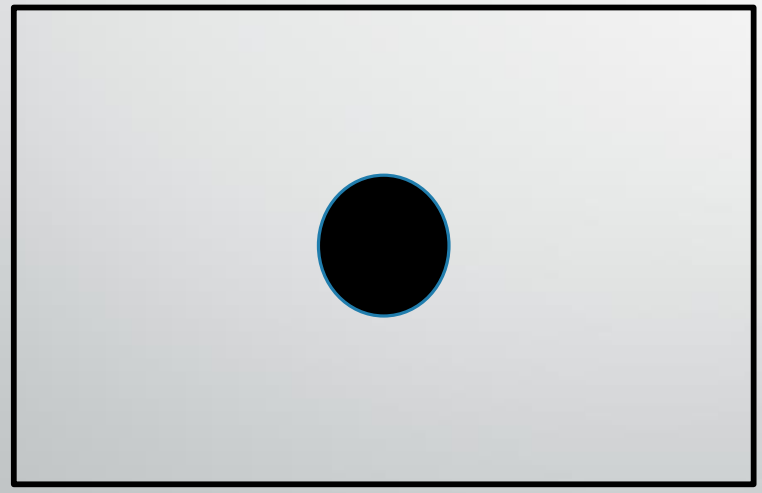

Fig 2. Initial state

2.2.2 Final state

- The final state is the state in which once reached, an object can never do a transition to another state.
- The final state may also mean that the object has actually been destroyed and can no longer be accessed.
- An object can have multiple stop states.
- The notation for a final/stop state is a bullseye, as shown in fig 3.

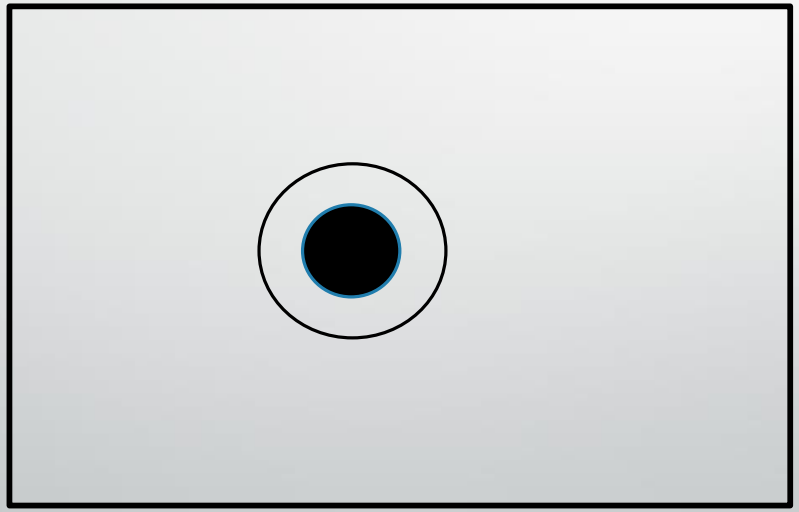

Fig 3. Final/stop state

2.3 Event

- Larman (2002) describes an event as "a significant or noteworthy occurrence".
- Stevens and Pooley (2006) describes it as "something done to an object, such as it being sent a message. An action is something that the object does, such as it sending a message".
- Ojo and Estevez (2005) describe it as "an occurrence of a stimulus that can trigger a state transition".
- An event may be:
  - the receipt of a signal, e.g. the reception of an exception
  - the receipt of a call, that is the invocation of an operation, for example, for changing the expiration date of a license
  - a change event
  - a time event

2.3 Event (cont'd)

- Another example of an event is simply a telephone receiver taken off it's hook (I know, in this day and age you simply ignore, divert to voice mail, or give a busy signal, right?)
- Note that an event on a statechart diagram corresponds to a message on a sequence diagram.
- The notation for an event is simply a pointed arrow in the direction the event is taking place; fig 4 shows the notation; fig 5 and fig 6 show examples of statechart diagrams.

Fig 4. Event

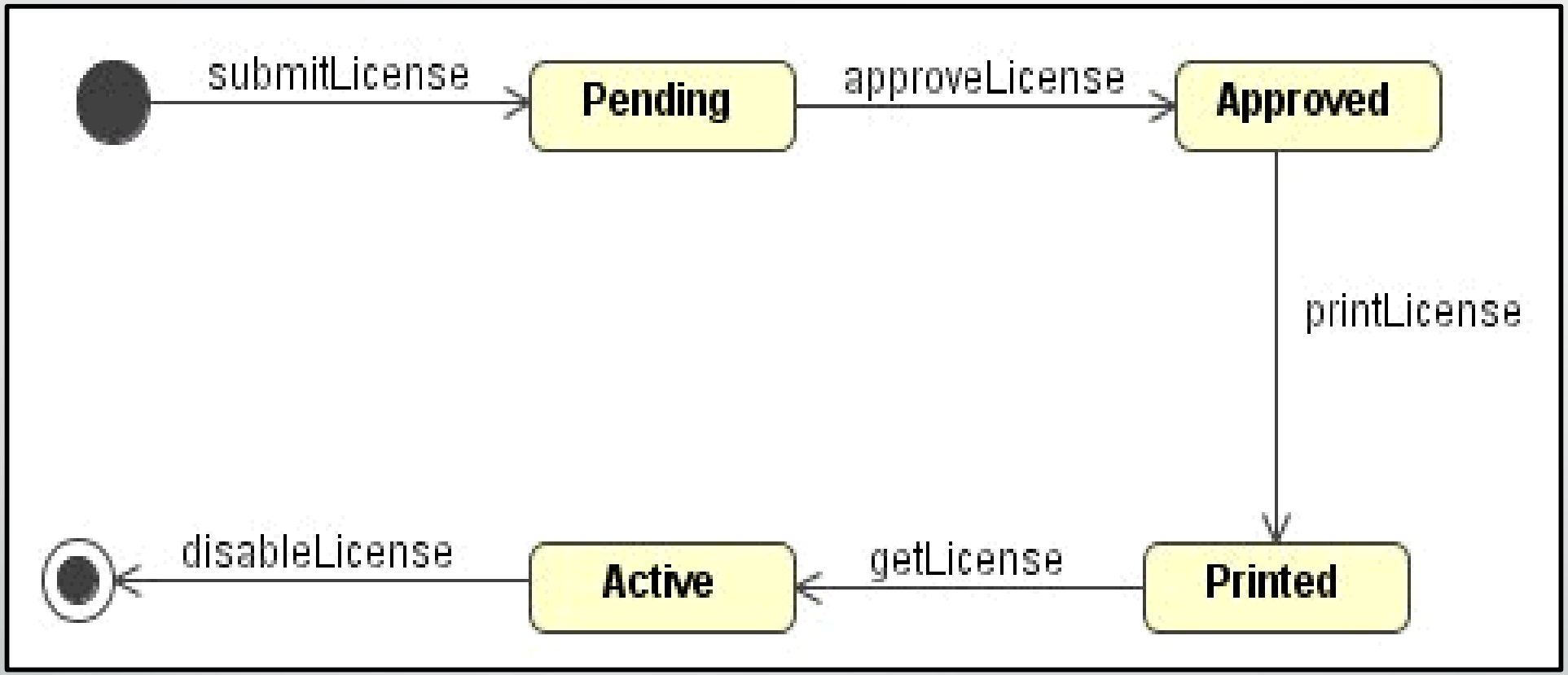

Fig 5 Statechart diagram example 1 (Ojo and Estevez, 2005)

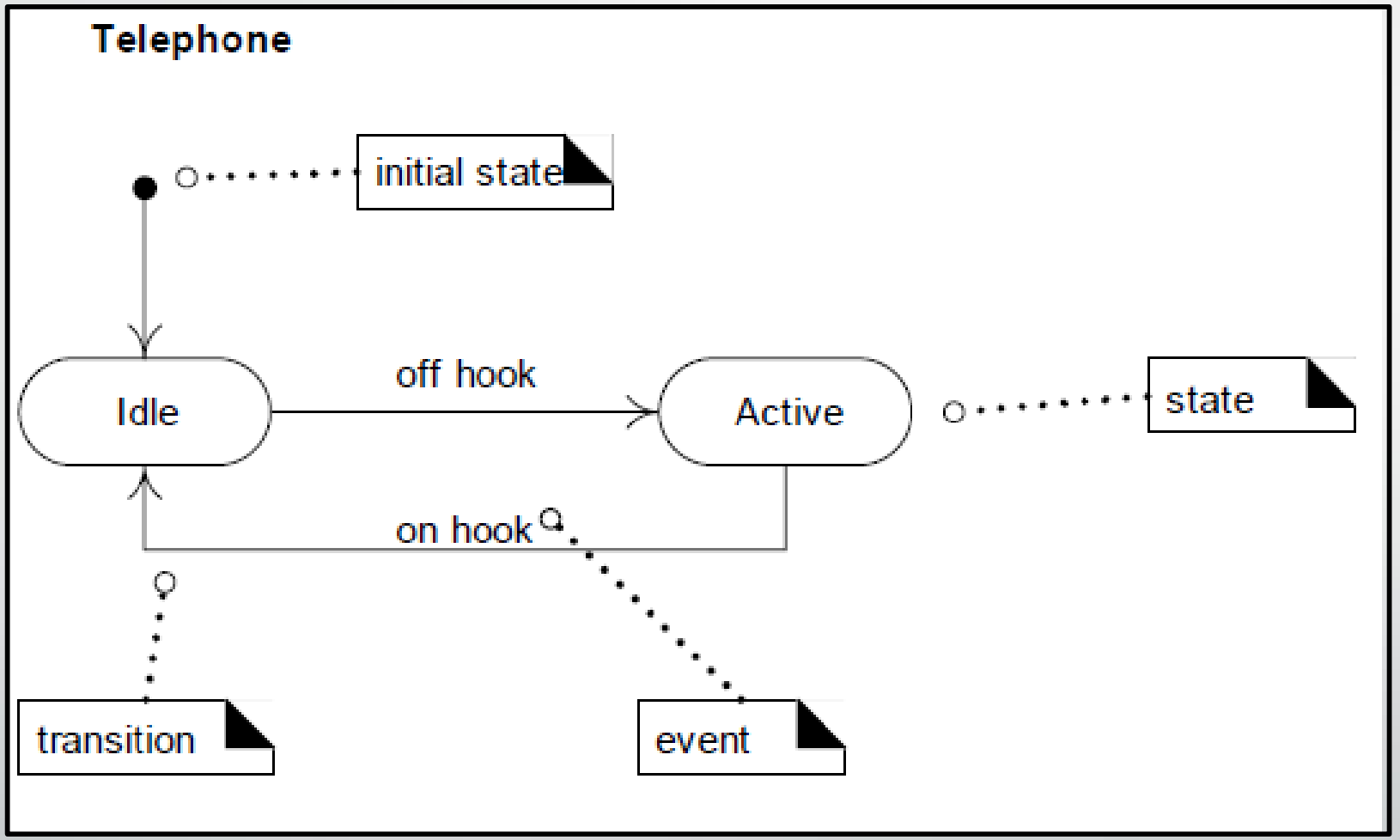

Fig 6. Statechart diagram for telephone (Larman, 2002)

2.4 Guard condition

- Typically, an event is received and responded unconditionally.
- When the receipt of an event is conditional, the test needed is called the guard condition.
- Fig 7 shows an example of an application of a guard condition.
- Fig 8 shows another example, which can be interpreted as follows: "On (*event 1* OR *event 2*) if (*guard condition* is *true*) then perform anAction and immediately enter state B.

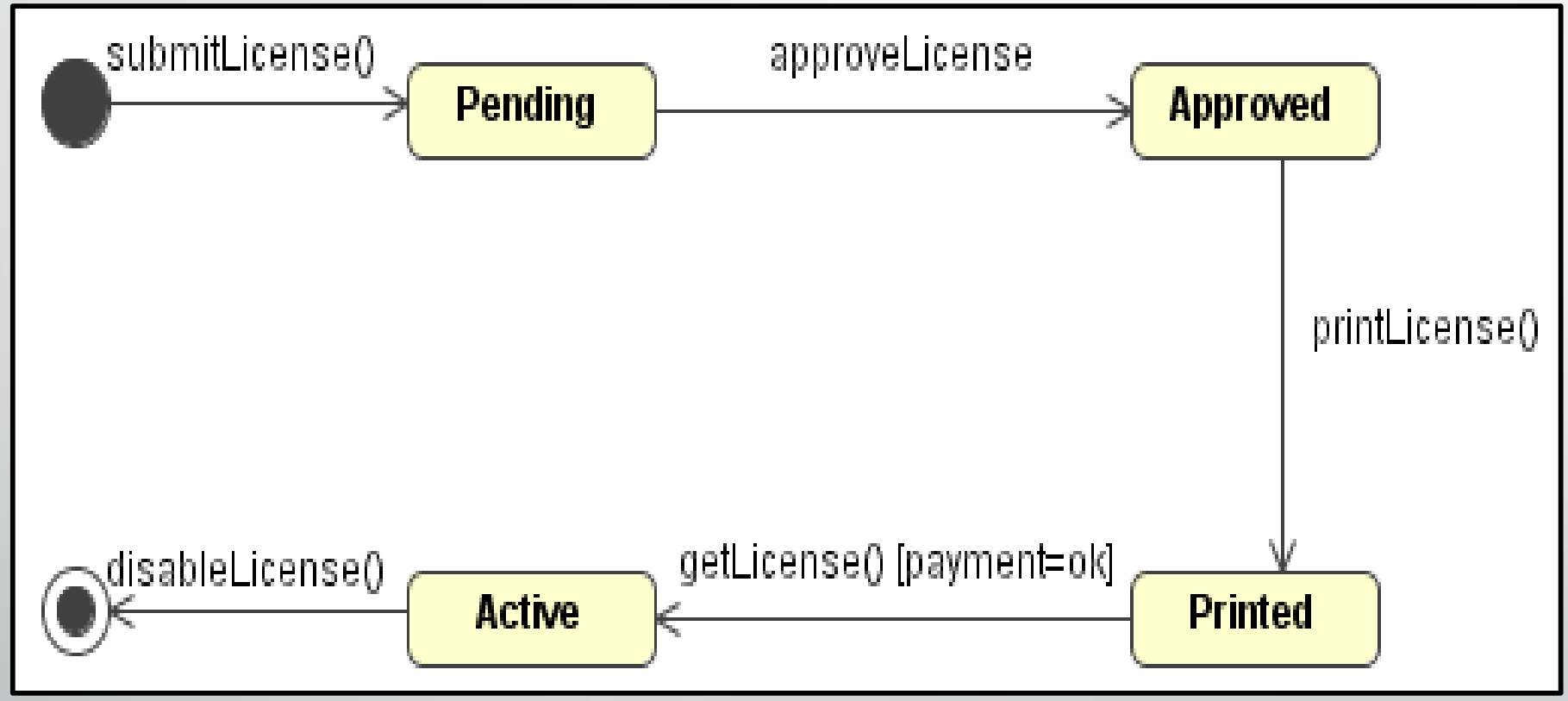

Fig 7. Guard condition example (Ojo and Estevez, 2005)

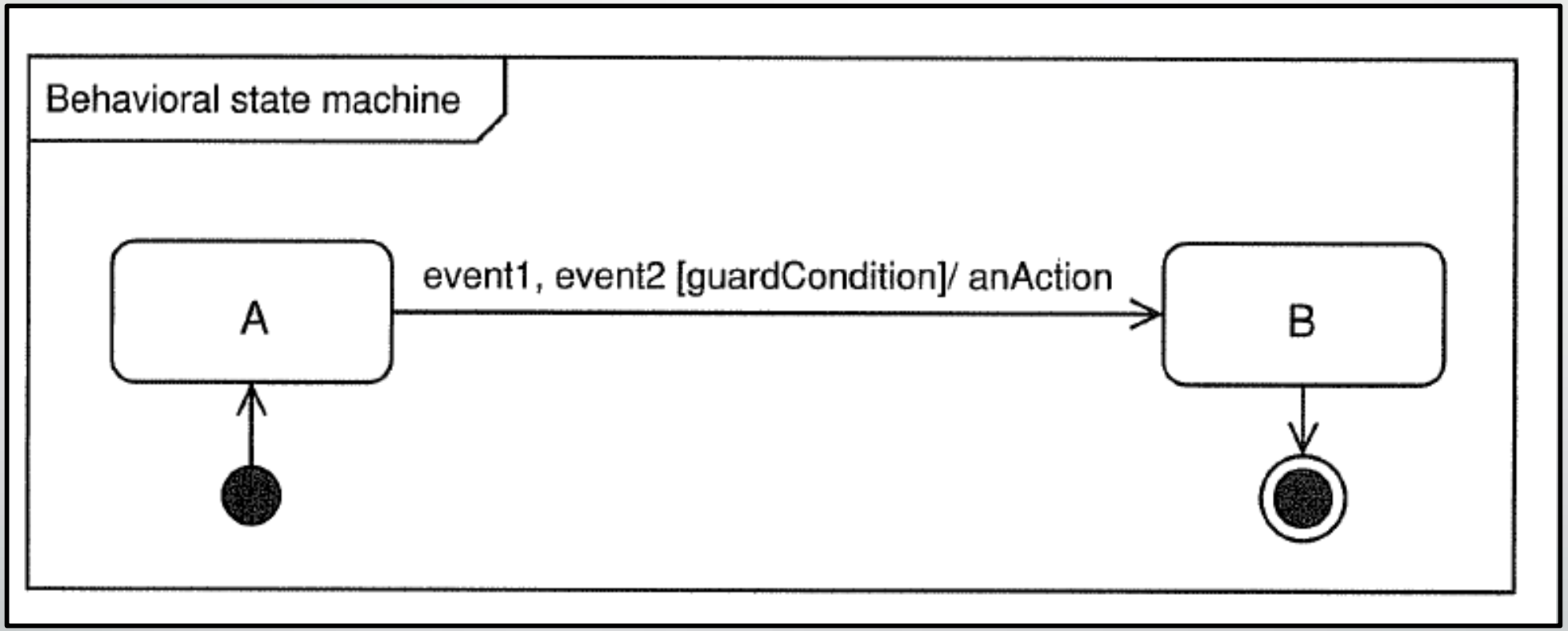

Fig 8. Guard condition example (Arlow and Neustadt, 2013)

2.5 Event action

- The response to an event has to explain how to change the attribute values that define the object's state.
- The behavior associated with an event is called action expression:
  - Part of a transition event – specifying the change from one state to another.
  - An atomic model of execution, referred to as run-to completion semantics.
- Fig 9 shows an example of event actions.

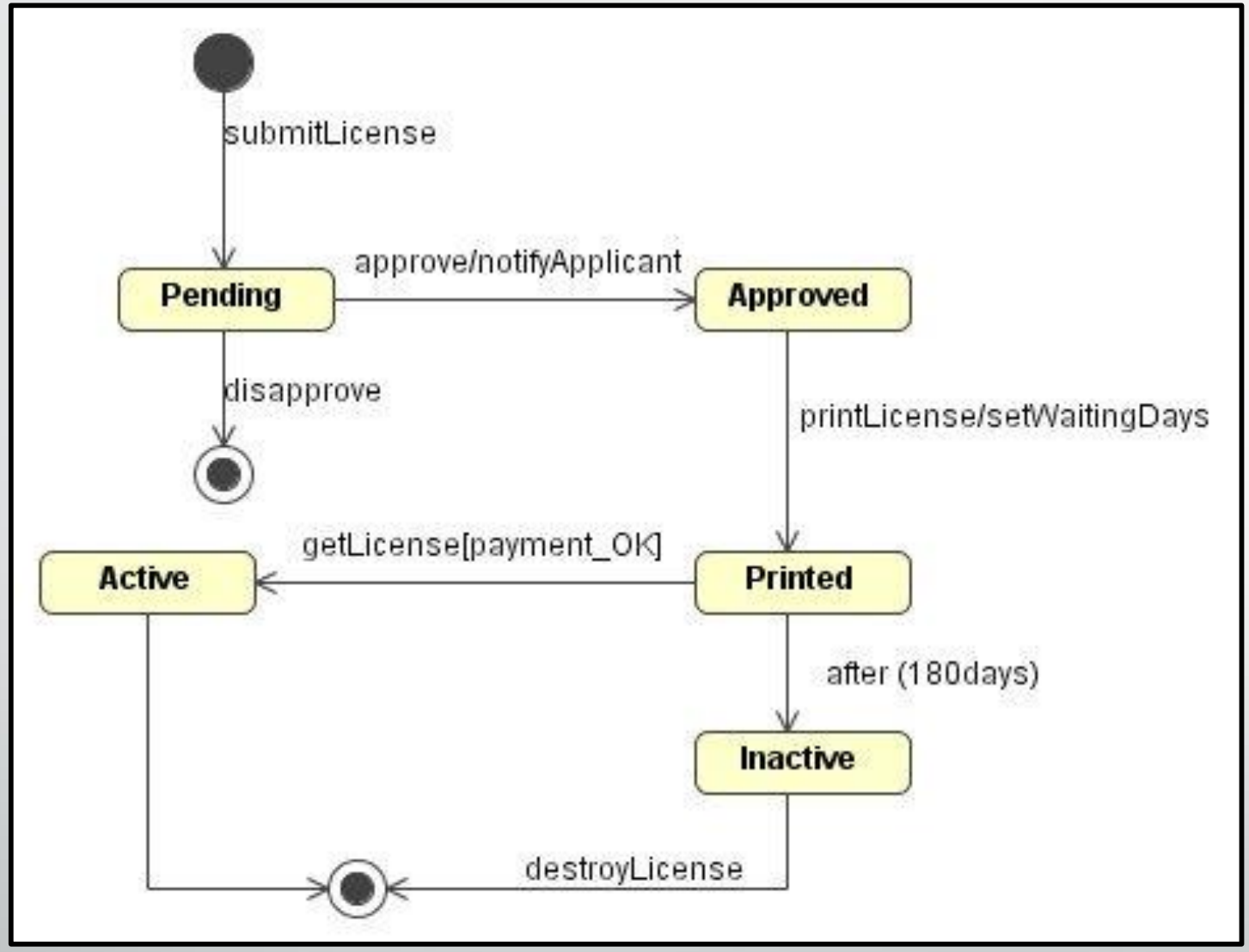

Fig 9. Event actions (Ojo and Estevez, 2005)

2.6 Event syntax

- The syntax for an event is as follows:
- event-name ‘(‘ [comma-separated-parameters-list] ‘) ‘ [‘[‘guard-condition’]’] / [action-expression]
- where:
- event-name - identifies the event
- parameters-list - data values passed with the event for use by the receiving object in its response to the event
- guard-condition - determines whether the receiving object should respond to the event
- action-expression - defines how the receiving object must respond to the event (Ojo and Estevez, 2005)

2.6 Event syntax (cont'd)

- Ojo and Estevez (2005) provide a simple example that demonstrates application of the event syntax as follows:
- approveLicense(License.Id) [req=ok] /setExistLicense(true)
- where:
- approveLicense is the event name
- License.Id is the event parameter
- the guard condition specifies that the req attribute must be OK for the receiving object to respond to the event.
- the action executed in the receiving object is a call to the method setExistLicense which sends true as its parameter

2.7 Transitions

- Larman (2002) defines a transition as "a relationship between two states that indicates that when an event occurs, the object moves from a prior state to a subsequent state…, for example when the event "off hook" occurs transition the telephone from the *idle* to *active* state."
- The transitions in a behavioral state machine are shown by arrows (external) or they may be nested (internal).
- Every transition has 3 optional elements (Arlow and Neustadt, 2013):
  - Zero or more events – occurrences that can trigger the transition, whether internal or external.
  - Zero or one guard condition – must evaluate to Boolean true for the transition to take place
  - Zero or more actions – these happen when the transition initiates.

2.7 Transitions (cont'd)

- Transitions in protocol state machines have a different syntax:
- There is no action (remember protocol state machines have no implementation)
- Guard conditions are specified by pre conditions and post conditions. Pre conditions are placed before the event while post conditions are placed after the slash.
- Further in both behavioral and protocol state machines if a transition has no event it is termed an automatic transmission; in such circumstances the transition fires/takes place when a guard condition is true, or precondition is true. (Arlow and Neustadt, 2013).
- What happens when you wish to connect transitions? There are two ways this can be achieved.

2.7.1 Connecting transitions - junction pseudo state

- Junction pseudo states represent places/points where transitions merge or branch.
- They are depicted as filled circles with one or more input transitions and also one or more output transitions.
- Fig 10 shows junction pseudo state for a loan class involved in the borrowing function of books in a library.
- In this case there is one junction with two inputs and one output.
- There are times when the junction pseudo state may have more than one output. In this case the output is dependent on only one guard condition evaluating to true among the outputs. Fig 11 shows such a situation for the same loan class in the instance that a loan needs to be extended.

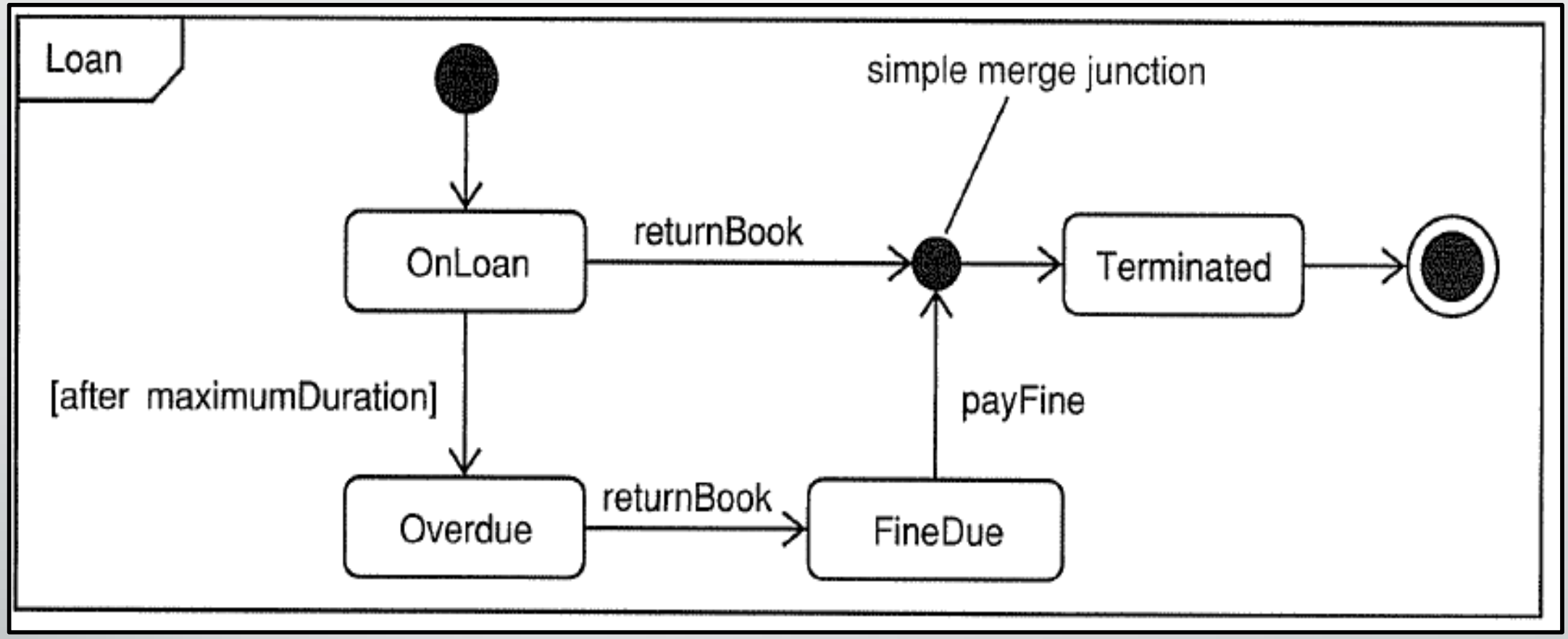

Fig 10. Junction pseudo state (Arlow and Neustadt, 2013)

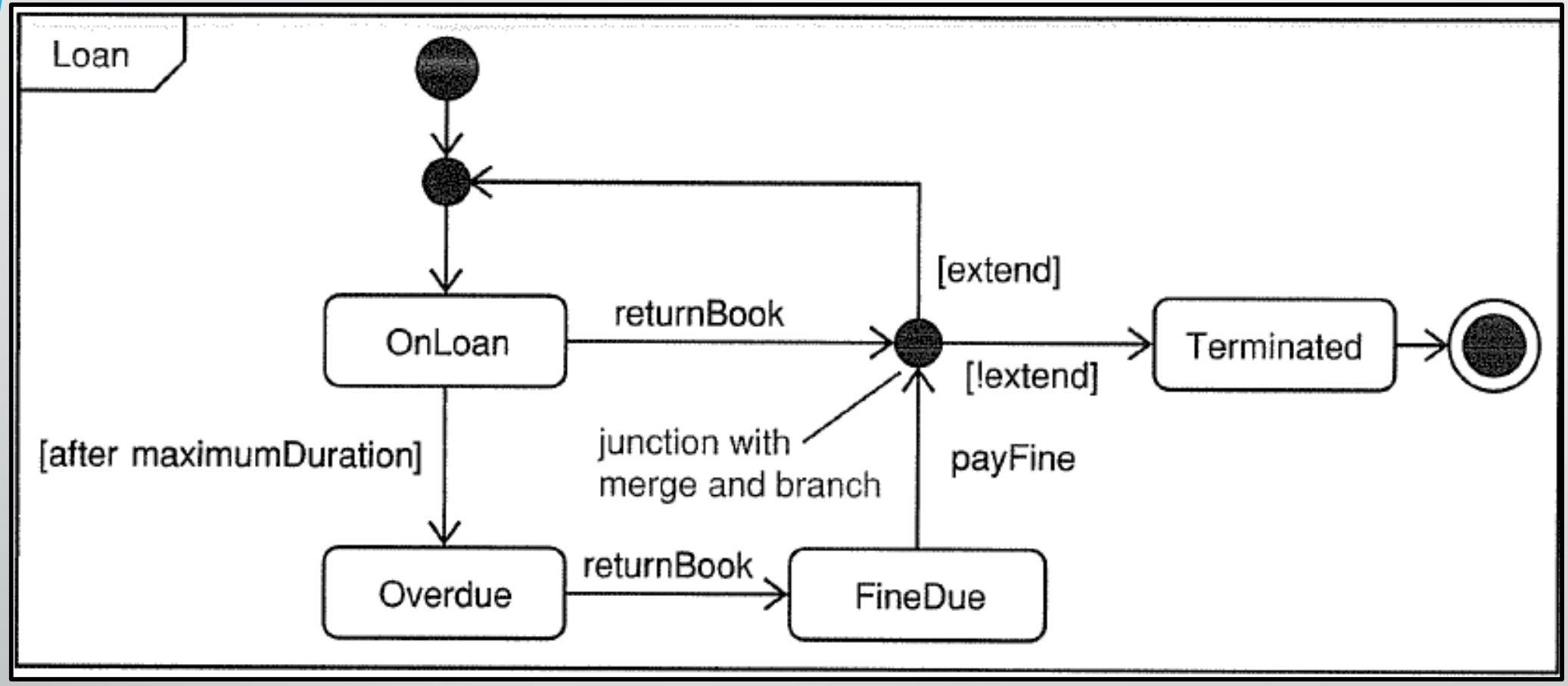

Fig 11. Junction pseudo state with more than one output (Arlow and Neustadt, 2013)

2.7.2 Branching transitions – the choice pseudo state

- These are used to show a branch without a merge.
- The choice allows you to direct the flow in the state machine outputs according to some set criteria or condition.
- This is different from when you are connecting transitions; in this case the desire is to branch rather than connect.
- Fig 12 shows an example of a branching transition scenario.

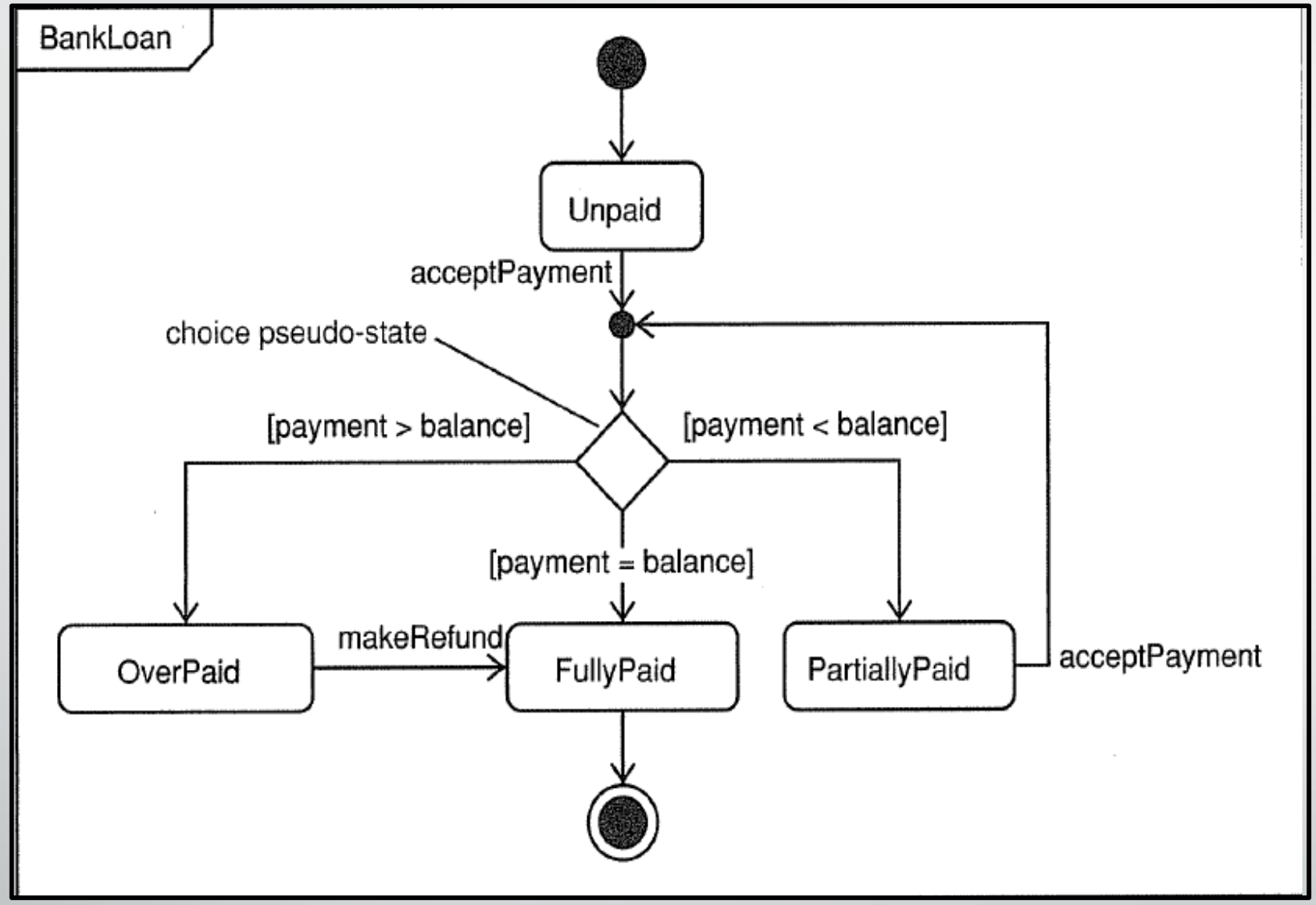

Fig 12. Branching pseudo state transition (Arlow and Neustadt, 2013)

2.7.3 Internal transition

- An internal transition is a way to handle events without leaving a state (or activity) and dispatching its exit or entry actions. An internal transition is shorthand for handling events without leaving a state and dispatching its exit or entry actions (docwiki.embarcadero.com)
- The UML notation is as follows:
  - uses the keyword internal transition followed by a slash and one event action
  - they are placed in the internal transitions compartment

2.7.4 Self transition

- A self – transition flow leaves the state dispatching any exit action(s), then reenters the state dispatching any entry action (s).
- An example is that a book is only borrowable if there is a copy of it in the library; if there is no copy then a copy must be returned in order for it to be borrowable as shown in fig 13:

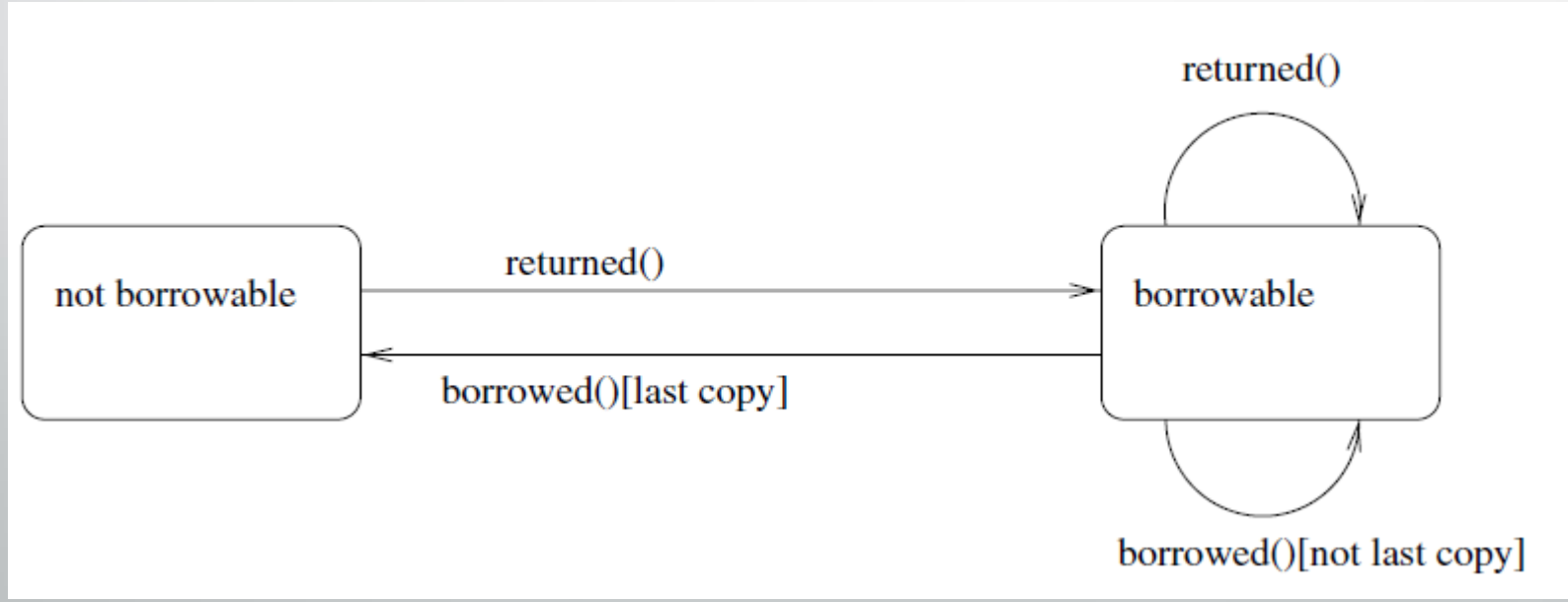

Fig 13. Borrowable state for book object (Stevens and Pooley, 2006)

Part 3

Complex states, entry and exit actions

3.1 Complex states

- The state icon can be expanded to model what the object can do while it is in a given state.
- The notation splits the state icon into two compartments:
  - name compartment and
  - internal transitions compartment
- The syntax that is used in the UML is shown in fig 14.
- As seen in fig 14, internal transitions compartments contains information about actions, activities and internal transitions specific to that state.

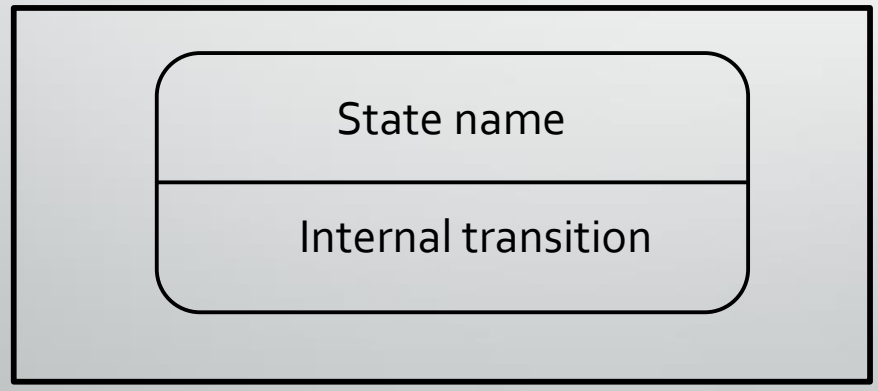

Fig 14. State name and internal transition

3.2 Entry actions

- More than one event can trigger a transition of an object into the same state.
- When the same action takes place in all events that goes into a state, the action may be written once as entry action. (Ojo and Estevez, 2005)
- Stevens and Pooley (2006) describes it thus: "we can show our intention directly, by writing the action inside the state as a reaction to the special event entry. There is implicitly an entry event every time the object enters a state, though we don't show or consider these events unless we want to associate actions with them."
- The notation in the UML is as follows:
- use the keyword entry followed by a slash and the actions to be performed every time the state is entered
- entry actions are part of internal transitions compartment
- Fig 15 shows an example of an entry action statechart diagram. It shows the scenario where the copy of a book can be returned to shelf, and then borrowed again by a user. In the entry event the start on the right shows initially the book has been returned on the shelf, then it is borrowed on loan, after which it is returned to the shelf. The two states entered by the copy class are, "on shelf", and "loan"; the actions performed are "borrowed" and "returned".

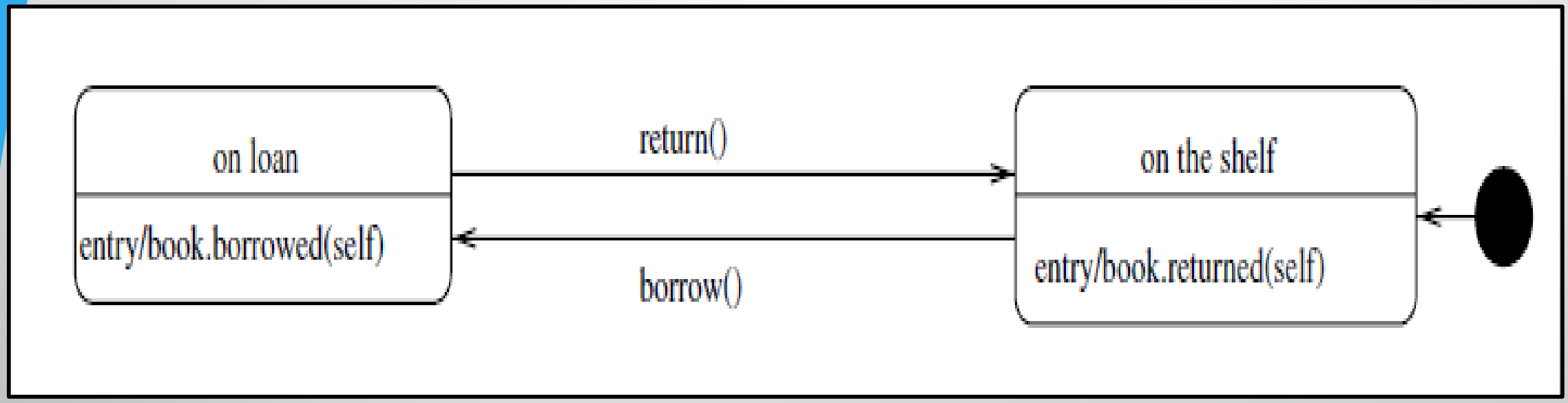

Fig 15. State diagram showing entry action (Stevens and Pooley, 2006)

3.3 Exit actions

- The same simplification can be used for actions associated with events that trigger a transition out of a state.
- Stevens and Pooley (2006) states that "we can show events which should happen whenever a given state is left by associating the action with an *exit* event in a state."
- They are called exit actions.
- The notation in the UML is as follows:
- use the keyword ***exit*** followed by a slash and the actions performed every time the state is exited
- exit actions are part of the internal transitions compartment
- Only when the action takes places every time the state is exited.

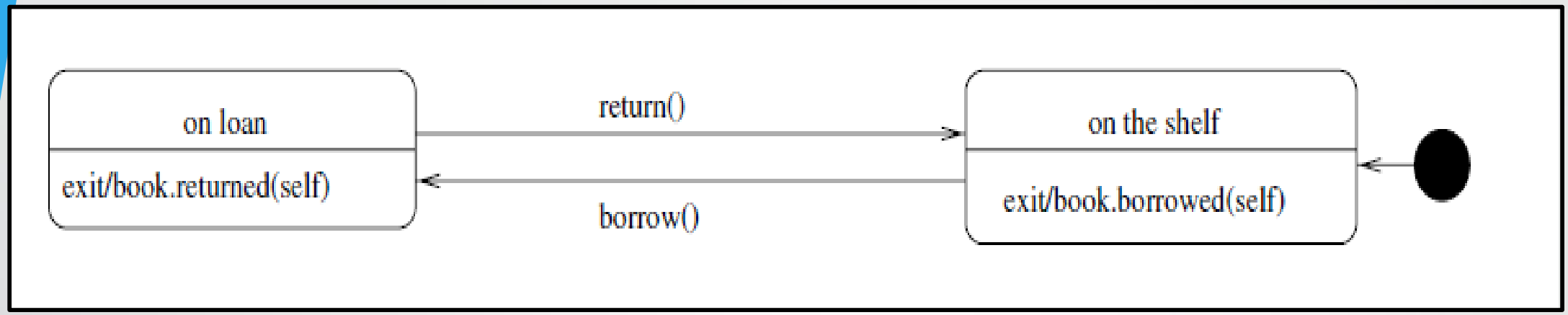

Fig 16. State diagram showing entry action (Stevens and Pooley, 2006)

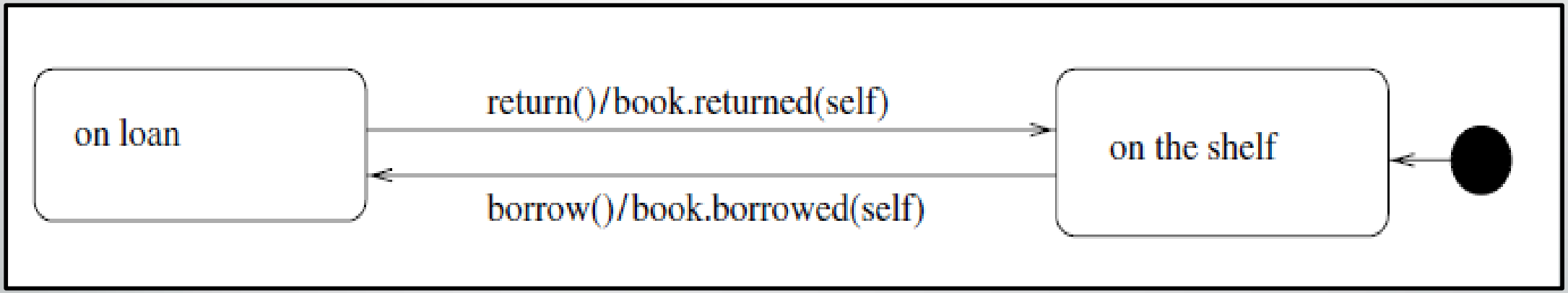

Fig 17. State diagram of class copy showing actions (Stevens and Pooley, 2006)

3.3 Exit actions (cont'd)

- Fig 16 shows the corresponding state diagram showing exit event. As can be rightfully deduced this is the opposite of fig 11 (entry action). The copy exits the "on shelf" state after it is borrowed, and similarly exits the "loan" state when it is returned.
- Fig 17 shows the state diagram without entry and exit actions; essentially you may notice that it is simply a combination of fig 15 and fig 16.
- Fig 18 presents another way to draw a state diagram showing the entry and exit actions. (Ojo and Estevez, 2005)

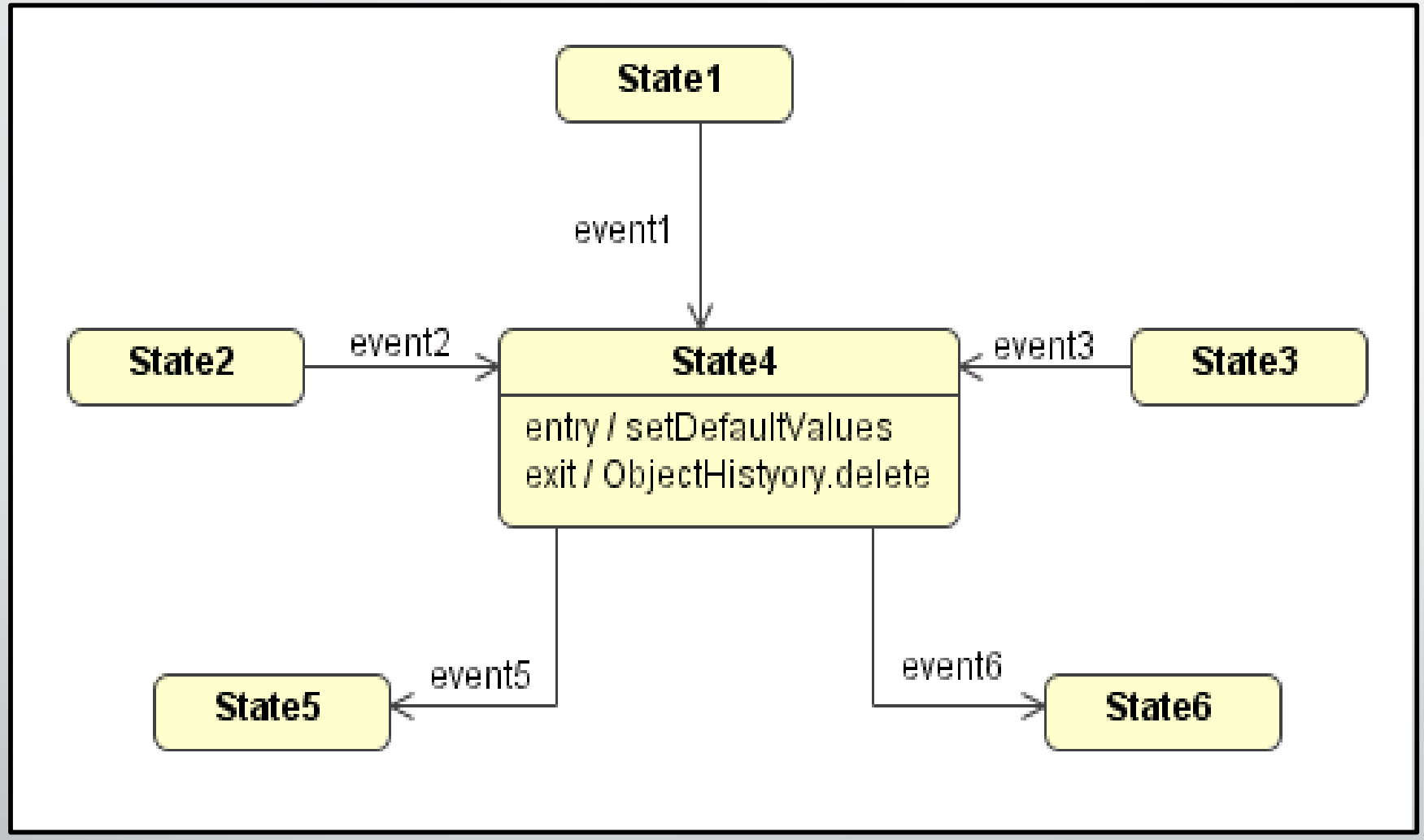

Fig 18. State diagram with entry and exit actions (Ojo and Estevez, 2005)

3.4 Activities

- Activities are the processes performed within a state.
- Activities may be interrupted because they do not affect the state of the object.
- The UML notation is as follows:
- use the do keyword followed by a slash and activities
- activities are placed in the internal transitions compartment.
- Fig 19 shows the notation for activity.

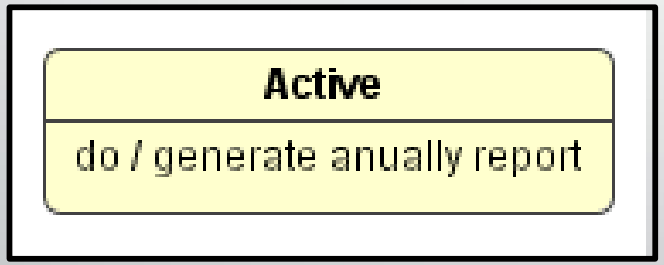

Fig 19. Activity notation (Ojo and Estevez, 2005)

3.4 Activities (cont'd)

- An activity should be performed:
- from the time the object enters the state, until:
  - either the object leaves the state or
  - the activity finishes.
- If an event produces a transition out of the activity state, the object must shut down properly and exit the state.

3.5 Other features

- Ojo and Estevez (2005) advise the following on state diagrams:
- Within the statechart diagram:
  - an object need not know who sent the message
  - an object is only responsible for how it responds to the event
- Focusing on the condition of the object and how it responds to the events, which object sends the message becomes irrelevant and the model is simplified
- The state of the object when it receives an event can affect the object's response
- event + state = response

Part 4

Advanced state machines

4.1 Composite states

- Composite states have been defined variously by diverse authors; sample these definitions:
- A composite state is a state that contains one or more other substates. It can be used to group states into logical compounds and thus make the statechart more comprehensible. (itemis.com)
- In a state machine diagram, a composite state is a state that has been decomposed into concurrent (representing "and" relationships) or mutually exclusive (representing "or" relationships) substates.(support.Microsoft.com)
- A composite state is a state that contains nested states, which in turn are organized into one or more state machines called submachines. Each submachine exists in its own region within the overall composite state (Arlow and Neustadt, 2013)
- Fig 20 shows an example of composite state.

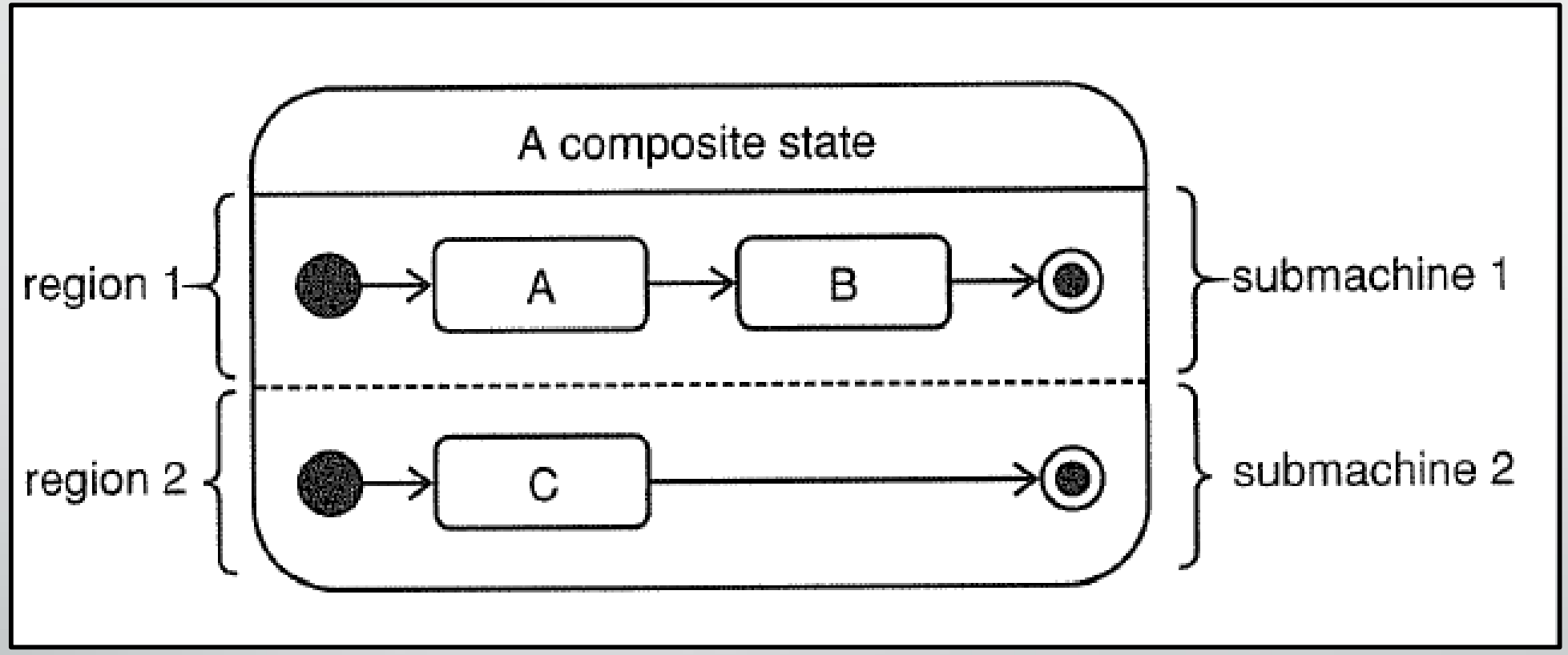

Fig 20. Composite state (Arlow and Neustadt, 2013)

4.1 Composite states (cont'd)

- The nested states inherit all the transitions of the container state.
- This means that if a container state has a transition then each of the nested states will also have this transition within them.
- The final pseudo state of a nested state applies only to its region; meaning that the different regions may terminate at different points in time.
- To stop the execution of a whole composite state use the terminate pseudo state, as shown in fig 21.
- Further we can also show a composite state leaving out the details of the regions using the UML notation shown in fig 22.
- Depending on the number of regions there are two types of composite states provided in the UML:
  - Simple composite state – one region only
  - Orthogonal composite state – more than one region

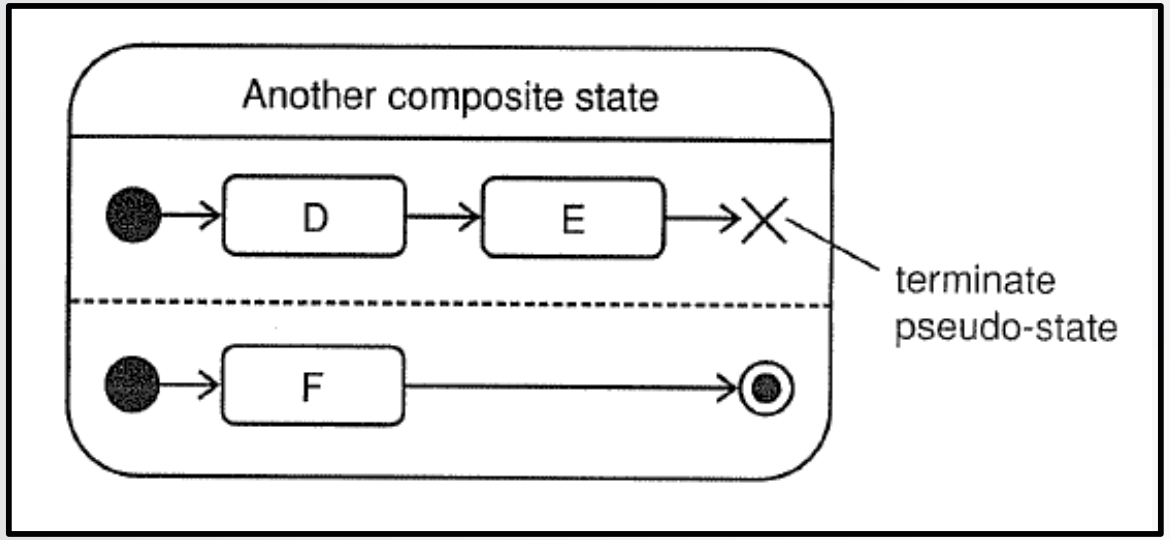

Fig 21. Terminate pseudo state (Arlow and Neustadt, 2013)

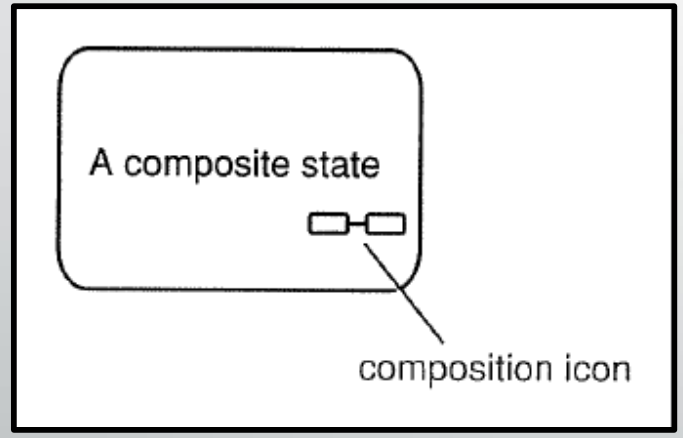

Fig 22. Composite state (Arlow and Neustadt, 2013)

4.1.1 Simple composite state

- In the UML a simple composite state is defined as a state that has substates.
- A state is allowed to have both states and submachines (defined later). A simple composite has only one region. An example of simple composite state machine is provided in fig 23.
- Note that in the UML a region is defined as an orthogonal part of either a composite state or a state machine.

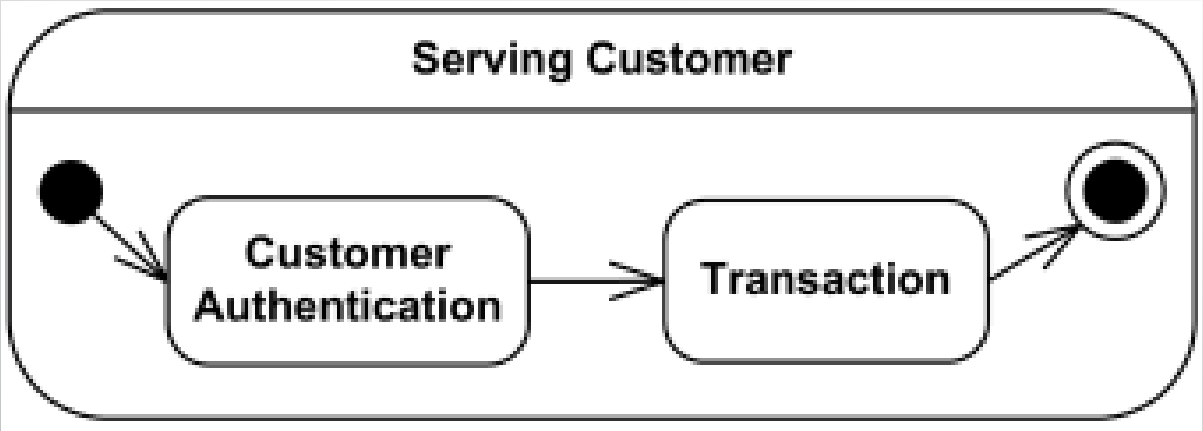

Fig 23. Serving customer has two substates (uml-diagrams.org)

4.1.2 Orthogonal composite state

- As defined earlier the orthogonal state has more than one region.
- Any state enclosed within a region of a composite state is called a **substate** of that composite state. It is called a **direct substate** when it is not contained by any other state; otherwise, it is referred to as an **indirect substate**.
- Each region of a composite state may have an initial pseudostate and a final state. A transition to the enclosing state represents a transition to the initial pseudostate in each region. A newly-created object takes its topmost default transitions, originating from the topmost initial pseudostates of each region. (uml-diagrams.org)
- Fig 24 shows a generic orthogonal composite state machine

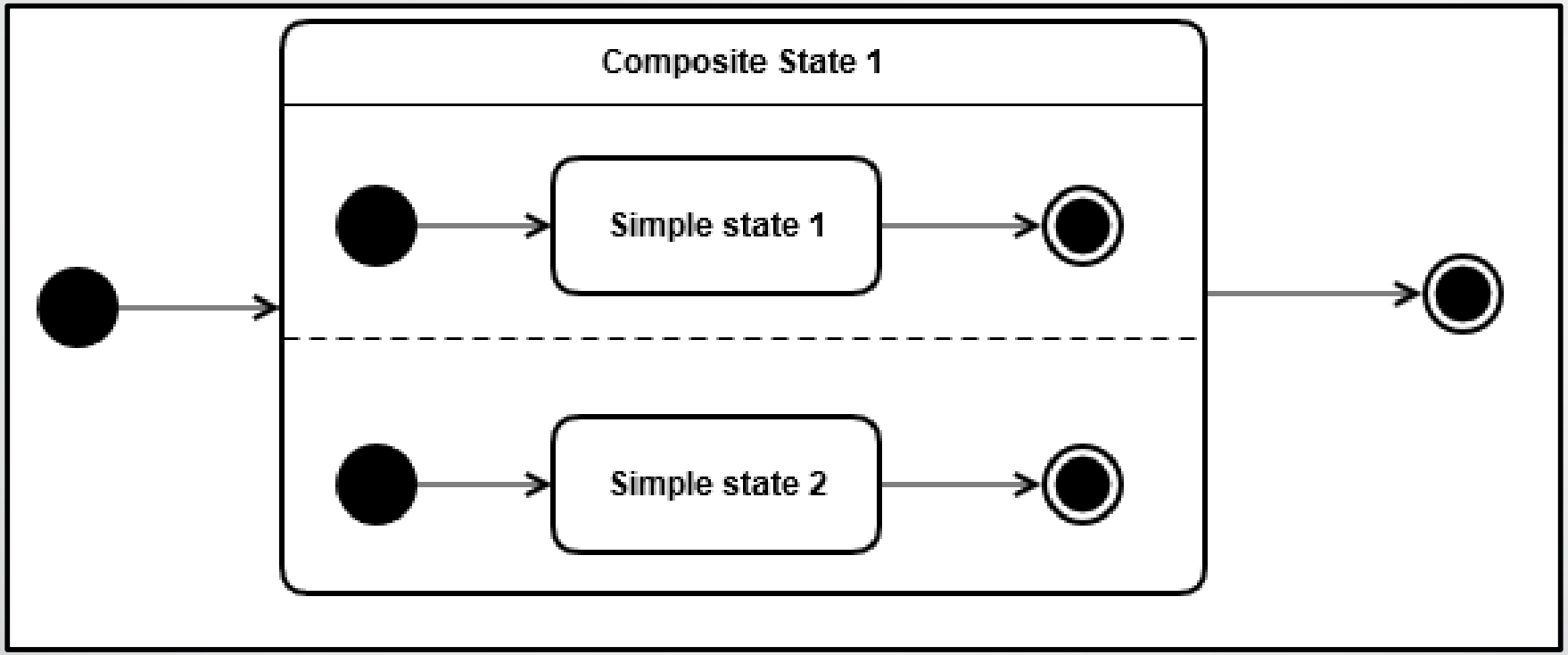

Fig 24. Generic orthogonal state machine (cybermedian.com)

4.1.2 Orthogonal composite state

- In an orthogonal composite state all sub states execute concurrently, implying an implicit fork. There are two ways to exit this:
- Both submachines finish (an implicit join)
- One of the submachines transitions to a state outside the superstate, normally via an exit pseudo state. There is no synchronization here (since there is no join) and the remaining submachines simply abort. (Arlow and Neustadt, 2013)
- Arlow and Neustadt (2013, pg 484) provide an example of a burglary system using this configuration.

4.2 Regions and submachines

- "Submachine state - A submachine state specifies the insertion of the specification of a submachine state machine. The state machine that contains the submachine state is called the containing state machine. The same state machine may be a submachine more than once in the context of a single containing state machine.
- A submachine state is semantically equivalent to a composite state. The regions of the submachine state machine are the regions of the composite state. The entry, exit, and behavior actions and internal transitions are defined as part of the state. Submachine state is a decomposition mechanism that allows factoring of common behaviors and their reuse.
- Region - A **region** is defined in UML 2.4 as an orthogonal part of either a **composite state** or a state machine. Region contains states and transitions.
- A composite state or state machine with regions is shown by tiling the graph region of the state/state machine using dashed lines to divide it into regions. Each region may have an optional name and contains the nested disjoint states and the transitions between these. The text compartments of the entire state are separated from the orthogonal regions by a solid line." (uml-diagrams.org)

4.3 Drawing the behavioral state diagram

- Dennis et al. (2015) suggest taking the following steps in developing a behavioral state diagram:
- Set context (usually based on a class)
- Identify object states (the various states the object will have over it's lifetime)
- Layout diagram (determine the sequence of the states that an object will pass through during its lifetime)
- Add transitions
- Validate (the final step is to validate the behavioral state machine by making sure that each state is reachable and that it is possible to leave all states except for final states. Obviously, if an identified state is not reachable, either a transition is missing or the state was identified in error. Only final states can be a dead end from the perspective of an object's life cycle.)

Summary

- The UML specifies two types of statechart diagrams: behavioral state machines and protocol state machines.
- Statechart diagrams are composed of states, events and complex states.
- A transition is a relationship between two states that indicates that when an event occurs, the object moves from a prior state to a subsequent state.
- Further in both behavioral and protocol state machines if a transition has no event it is termed an automatic transmission; in such circumstances the transition fires/takes place when a guard condition is true, or precondition is true.
- Junction pseudo states represent places/points where transitions merge or branch.
- Branching transitions are used to show a branch without a merge.
- An internal transition is a way to handle events without leaving a state (or activity) and dispatching its exit or entry actions
- Depending on the number of regions there are two types of composite states provided in the UML: simple composite state and orthogonal composite state.

References

- Arlow, J., & Neustadt, I. (2013). *Uml 2 and the unified process: Practical object-oriented analysis and Design* (Second). Addison-Wesley.
- Dennis, A., Wixom, B. H., Tegarden, D. P., & Seeman, E. (2015). *System analysis & design: An object-oriented approach with Uml*. Wiley.
- Fakhroutdinov, K. *State Machine Diagrams*. https://www.uml-diagrams.org/state-machine-diagrams.html.
- Fakhroutdinov, K. "State Machine Diagrams." *UML Graphical Notation Overview, Examples, and Reference.*, https://www.uml-diagrams.org/state-machine-diagrams.html#submachine-state.
- Larman, C. (2002). *Applying Uml and patterns: An introduction to object-oriented analysis and design and the Unified Process*. Prentice Hall.
- Ojo, A., & Estevez, E. (2005). (rep.). *Object-Oriented Analysis and Design with UML - Training Course* (Vol. 1). In E-Macao Report 19.
- Stevens, P., & Pooley, R. (2006). *Using UML: Software Engineering with Objects and Components*. Pearson Educaion Limited.
- "What Is a Composite State in a UML State Machine Diagram?" *Cybermedian*, 3 Mar. 2022, https://www.cybermedian.com/what-composite-states-in-uml-state-machine-diagram/.