CS 425

UML Activity Diagrams & State Charts

October 31, 2024

From Chapters 14, 21, and 22 [Jim Arlow and Ila Neustadt, UML 2 and the Unified Process, Addison-Wesley 2005]

University of Nevada, Reno
Department of Computer Science & Engineering

Outline

Part 1 - Activity diagrams

- Introduction
- Activities
- Nodes
  - Action nodes
  - Control nodes
  - Object nodes
- Activity parameters

Introduction: What are activity diagrams?

Activity diagrams:

- A form of "object-oriented flowcharts"
- Attached to modeling elements to describe behavior
- Typically attached to use cases, classes, operations, components, and interfaces
- Can also be used to model business processes and workflows

Introduction: Where are activity diagrams used?

Commonly used in:

- Analysis
  - To model the flow of a use case
  - To model the flow between use cases
- Design
  - To model details of an operation
  - To model details of an algorithm
- Business modeling
  - To model a business process
- As always in modeling, it is important to keep them simple and understandable by their intended audience

Activities *****

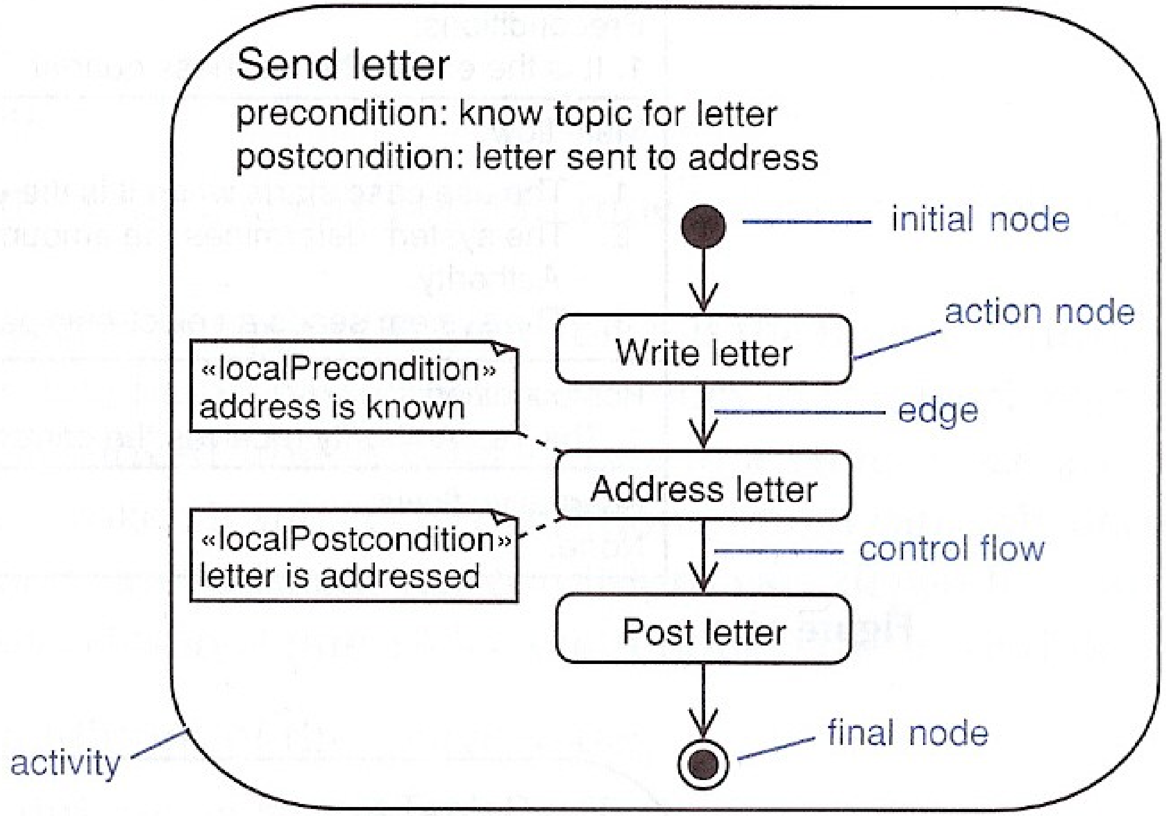
- *Activity diagrams* are networks of *nodes* connected by *edges*
- Nodes
  - Action nodes – atomic units of work within the activity
  - Control nodes – control the flow through the activity
  - Object nodes – represent objects used in the activity
- Edges
  - Control flows – depict the flow of control through activity
  - Object flows – depict the flow of objects through activity

* Activities ****

- *Activities* and *actions* can have pre- and post-conditions
- *Tokens* (part of semantics but not shown graphically) abstractly flow in the network and can represent:
  - The flow of control
  - An object
  - Some data
- A token moves from a source node to a target node across an edge depending on:
  - Source node post-conditions
  - Edge guard conditions
  - Target node preconditions

** Activities ***

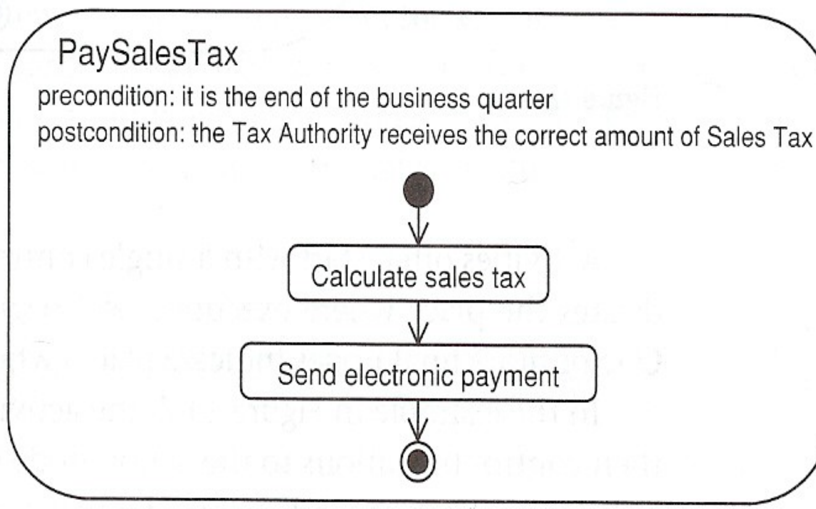
- Example of an activity ("send letter"), Fig. 14.2 [Arlow & Neustadt 2005]



*** Activities **

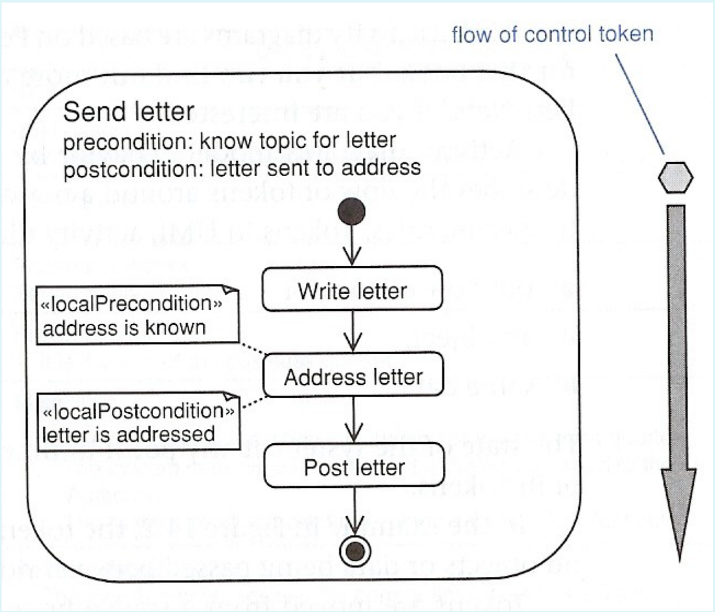
- Activity diagrams can model use cases as a series of actions.

Use case: PaySalesTax
ID: 1
Brief description: Pay Sales Tax to the Tax Authority at the end of the business quarter.
Primary actors: Time
Secondary actors: TaxAuthority
Preconditions: 1. It is the end of the business quarter.
Main flow: 1. The use case starts when it is the end of the business quarter. 2. The system determines the amount of Sales Tax owed to the Tax Authority. 3. The system sends an electronic payment to the Tax Authority.
Postconditions: 1. The Tax Authority receives the correct amount of Sales Tax.
Alternative flows: None.



**** Activities *

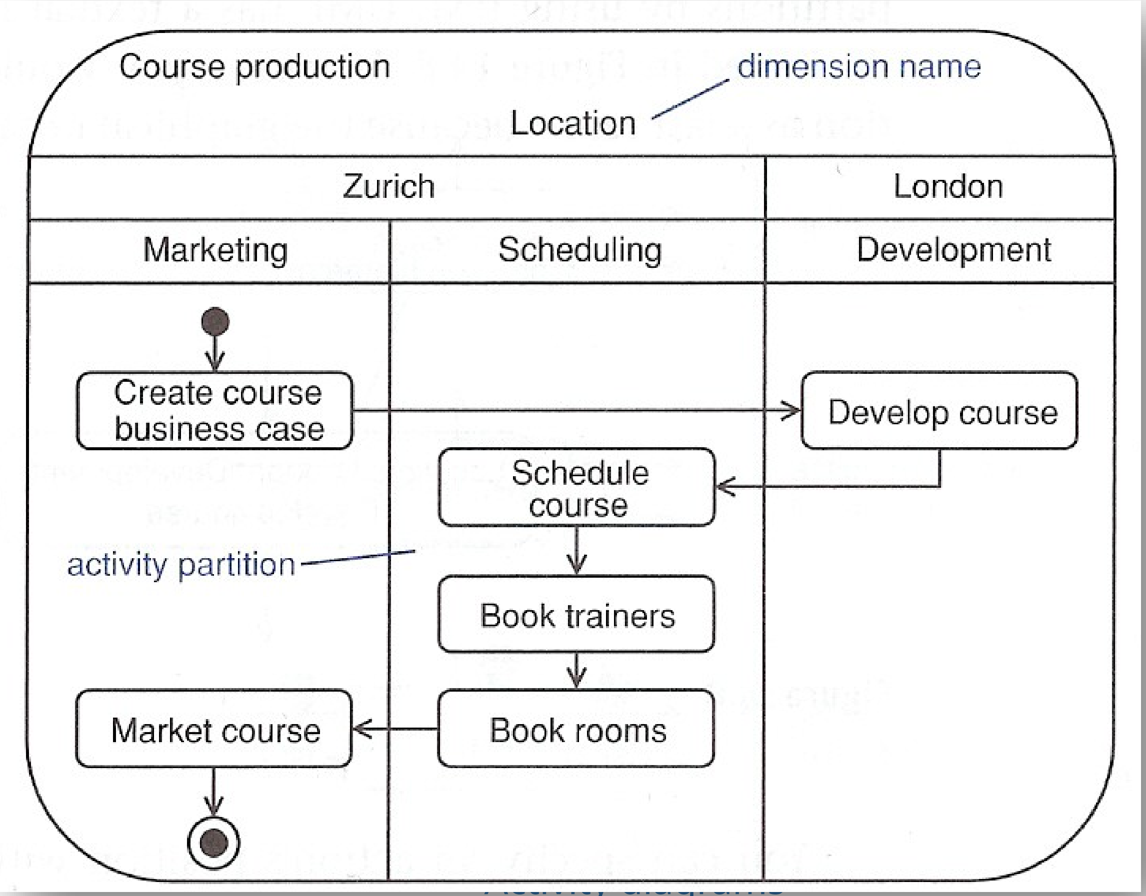
- Activity diagrams have semantics based on Petri Nets
- They model behavior using the token game
- Tokens move through the network subject to conditions
- Object nodes represent objects flowing around the system
- Example of flow of control token



Activity diagrams

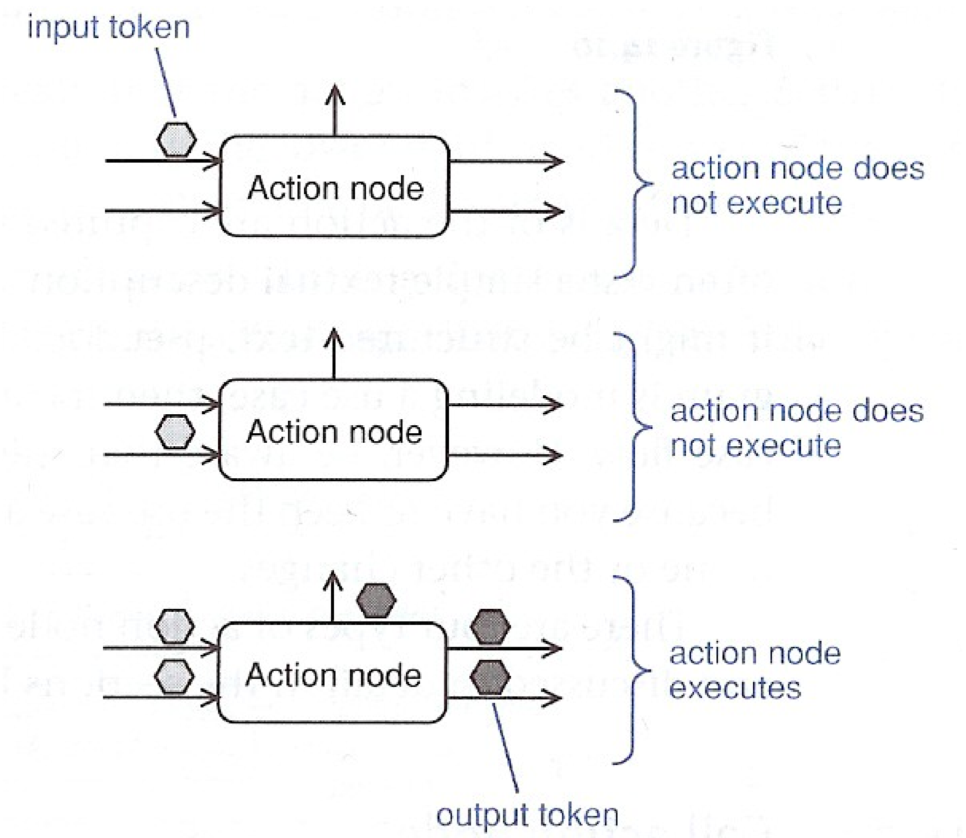
***** Activities

- Activity diagrams can be divided in *partitions* (*swimlanes*) using vertical, horizontal, or curved lines.



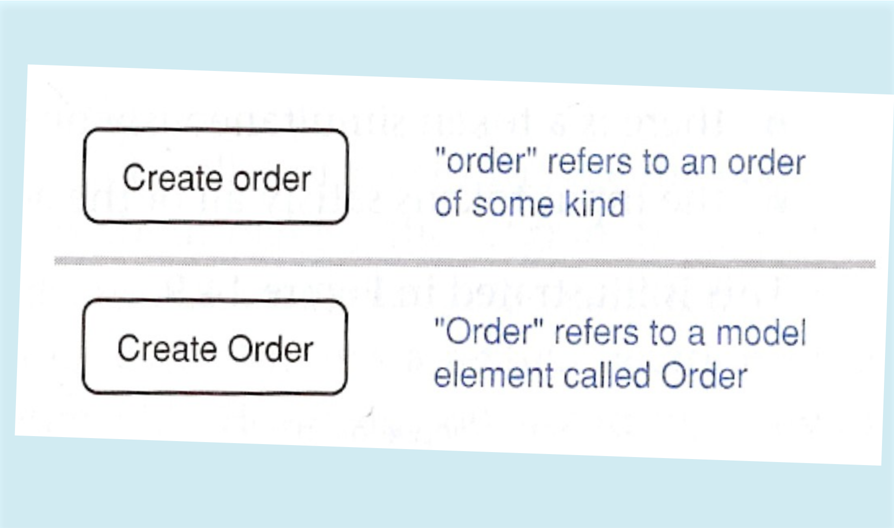
Action nodes ****

- Action nodes execute when:
  - There are tokens present at all their input nodes AND
  - The input tokens satisfy all action node's local preconditions



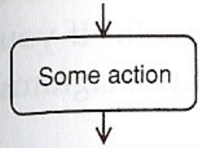
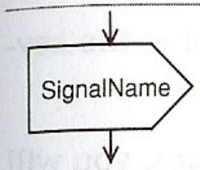
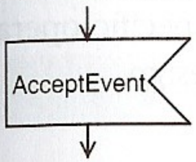
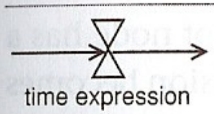
* Action nodes ***

- After execution, the local post-conditions are checked; if all are satisfied, the node simultaneously offers tokens to all its output edges (this is an implicit fork that may give rise to many flows)



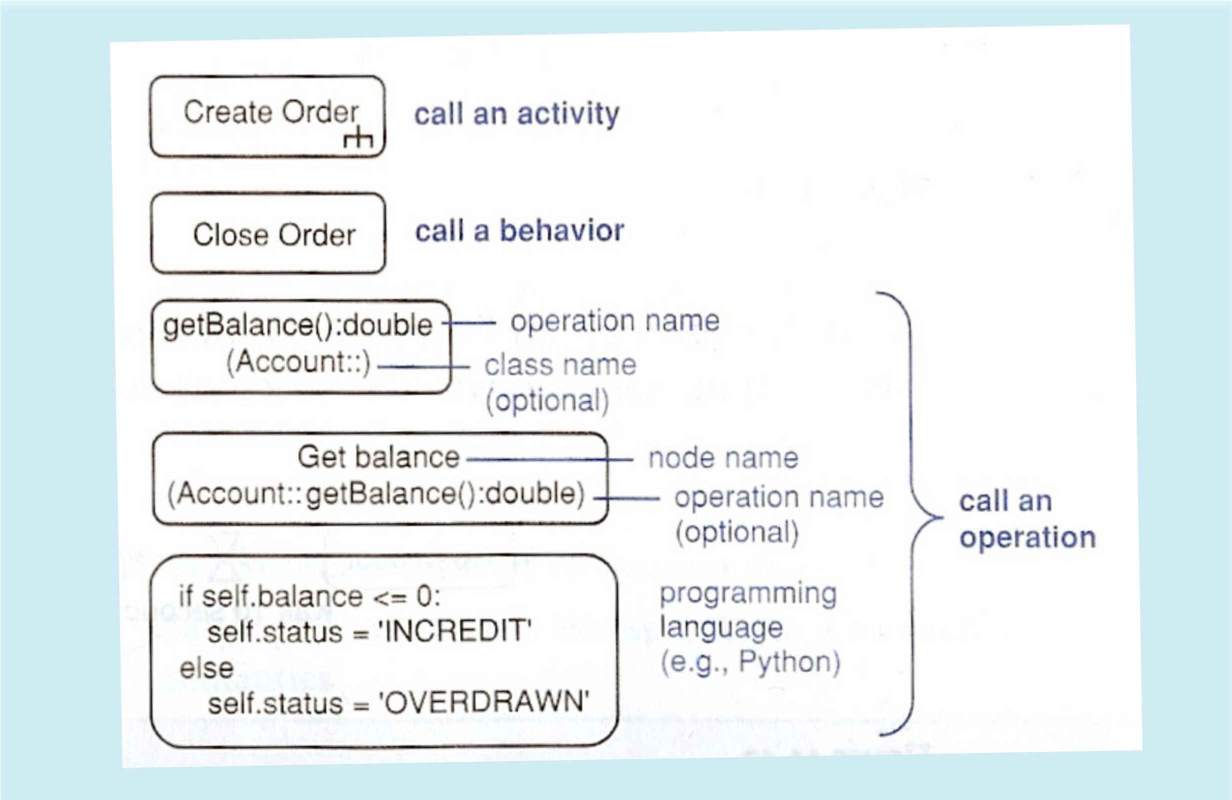
** Action nodes **

- Types of *action nodes*, Table. 14.1 [Arlow & Neustadt 2005]

Syntax	Name	Semantics	Section
	Call action node	Invokes an activity, behavior, or operation	14.7.1
	Send signal	Send signal action – sends a signal asynchronously (the sender *does not* wait for confirmation of signal receipt) It may accept input parameters to create the signal	15.6
	Accept event action node	Accepts an event – waits for events detected by its owning object and offers the event on its output edge Is enabled when it gets a token on its input edge If there is *no* input edge, it starts when its containing activity starts and is always enabled	15.6
	Accept time event action node	Accepts a time event – responds to time Generates time events according to its time expression	14.7.2

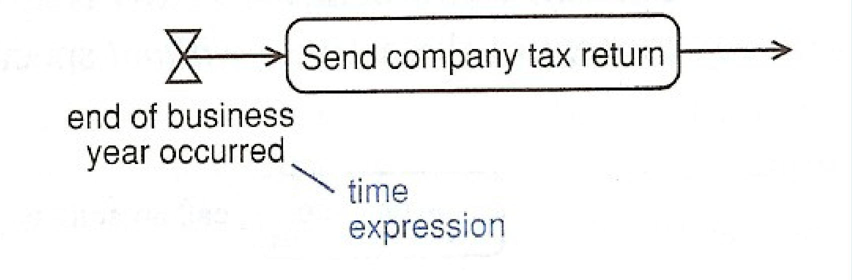
*** Action nodes *

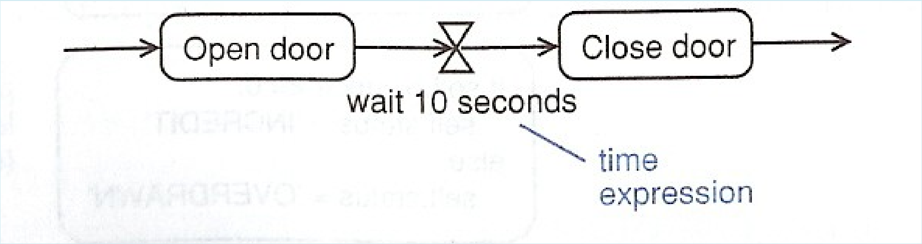
- A *call action node* invokes an activity, behavior, or operation



**** Action nodes

- An *accept time event* action node responds to time





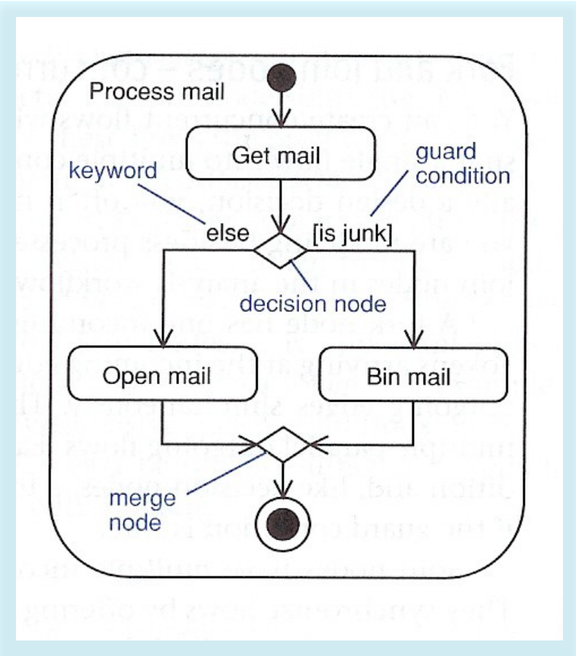
Control nodes **

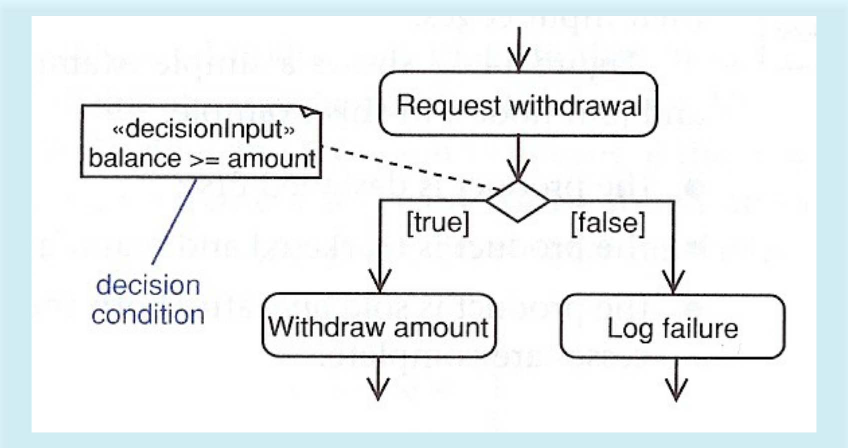
Control nodes manage the flow of control within an activity
Table 14.2 [Arlow & Neustadt 2005] shows the types of control nodes

Syntax	Name	Semantics		Section
	Initial node	Indicates where the flow starts when an activity is invoked		14.8.1
	Activity final node	Terminates an activity	Final nodes	14.8.1
	Flow final node	Terminates a specific flow within an activity – the other flows are unaffected	Final nodes	14.8.1
«decisionInput» decision condition	Decision node	The output edge whose guard condition is true is traversed May optionally have a «decisionInput»		14.8.2
	Merge node	Copies input tokens to its single output edge		14.8.2
	Fork node	Splits the flow into multiple concurrent flows		14.8.3
{join spec}	Join node	Synchronizes multiple concurrent flows May optionally have a join specification to modify its semantics		14.8.3

* Control nodes *

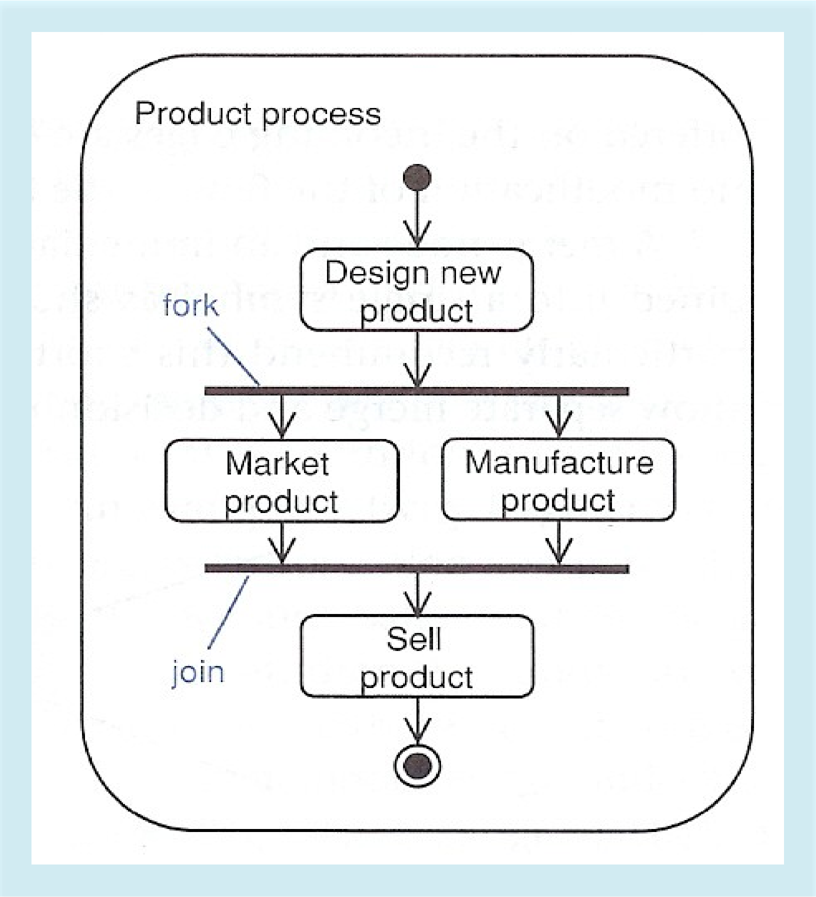
- Examples of *decision* and *merge* nodes





** Control nodes

- Examples of *join* and *fork* nodes



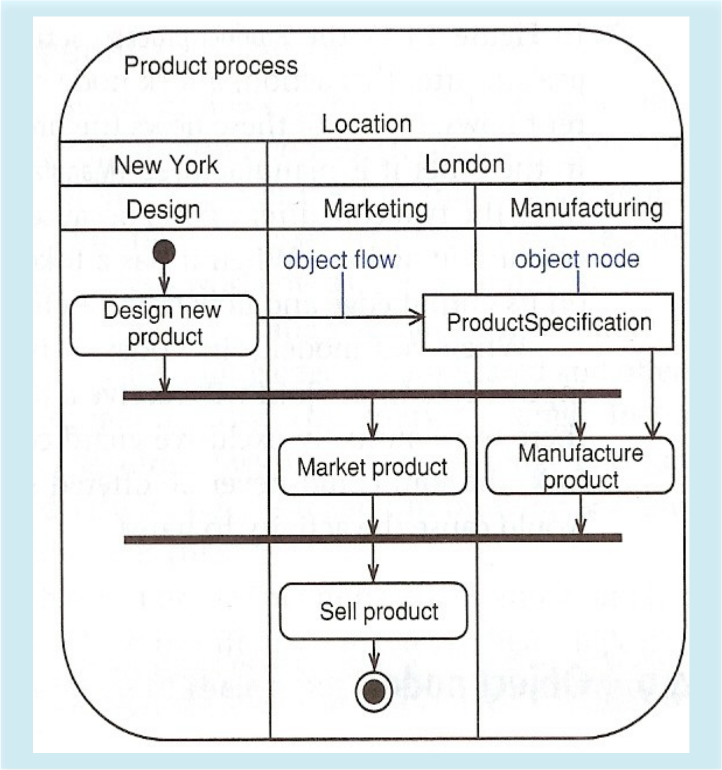
Activity diagrams

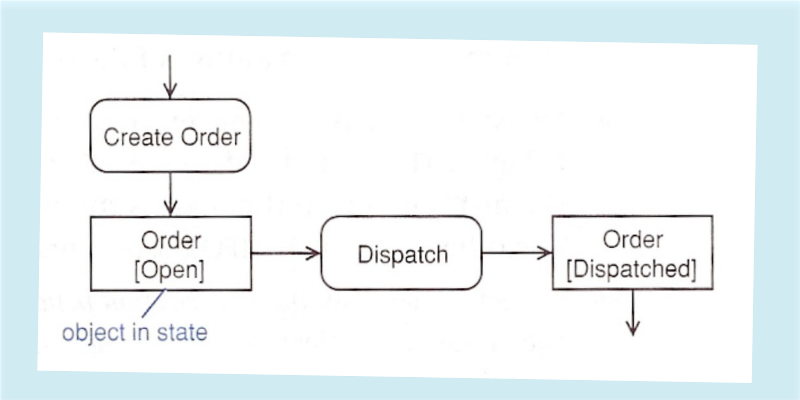
Object nodes *

- *Object nodes* indicate that instances of a particular classifier are available at a specific point in the activity
- They are labeled with the name of the classifier and represent instances of that classifier or its subclasses
- The input and output edges are *object flows*
- The objects are created and consumed by action nodes
- When an object node receives an object token on one of its input edges, it offers this token to all its output edges, which compete for the token.

* Object nodes

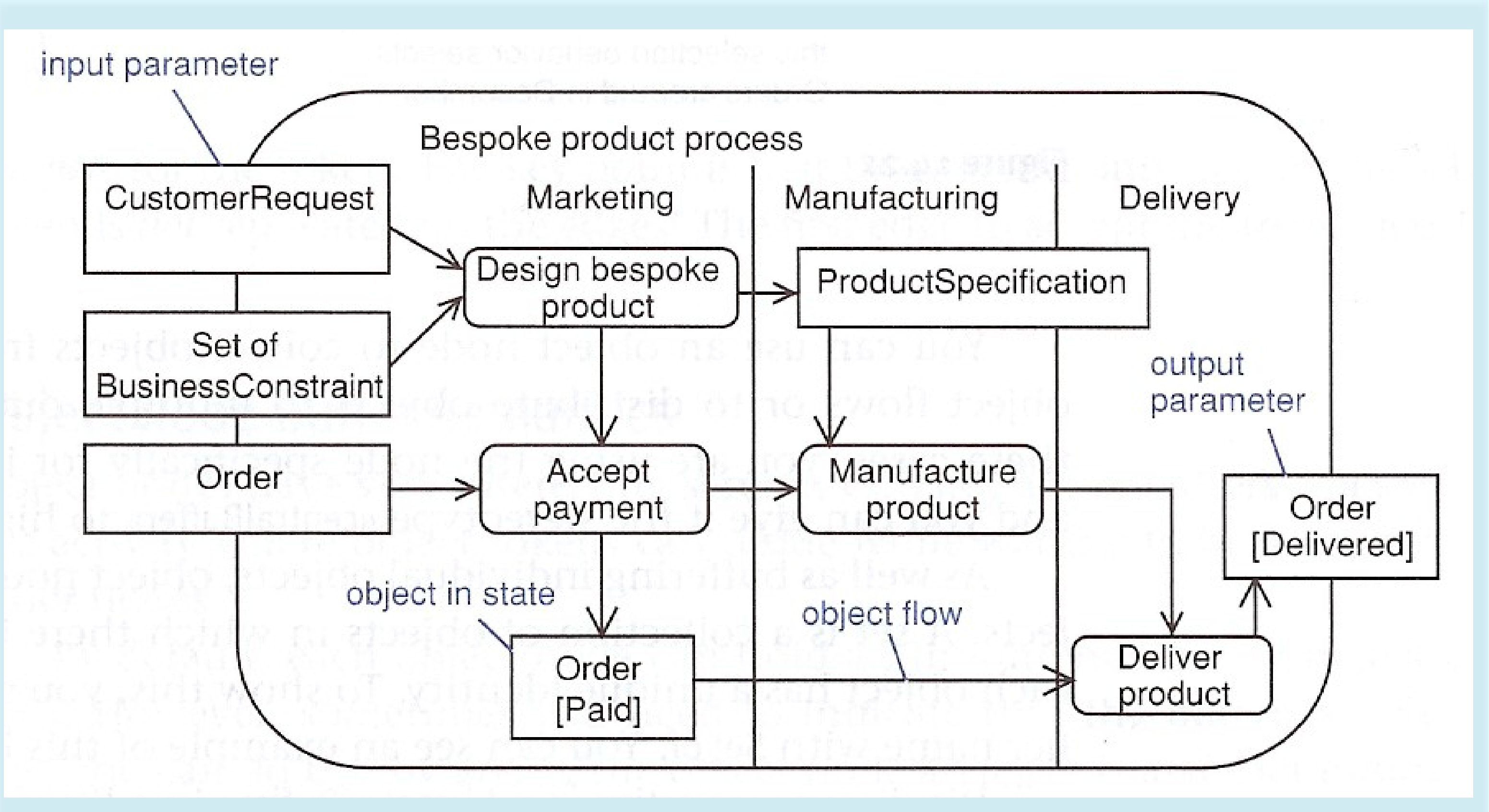
- Examples of using object nodes. Note that object nodes can represent objects in particular states.





Activity parameters

- Activities can have object nodes to provide inputs and outputs



Part 2 – State Charts

From Chapter 21: State Machines &
Chapter 22: Advanced State Machines (partial)

[Arlow and Neustadt 2005]

Outline

- State machines
  - Introduction
  - State machine diagrams
  - States
  - Transitions
  - Events
- Advanced state machines
  - Composite states
    - Simple
    - Orthogonal
  - History

Introduction

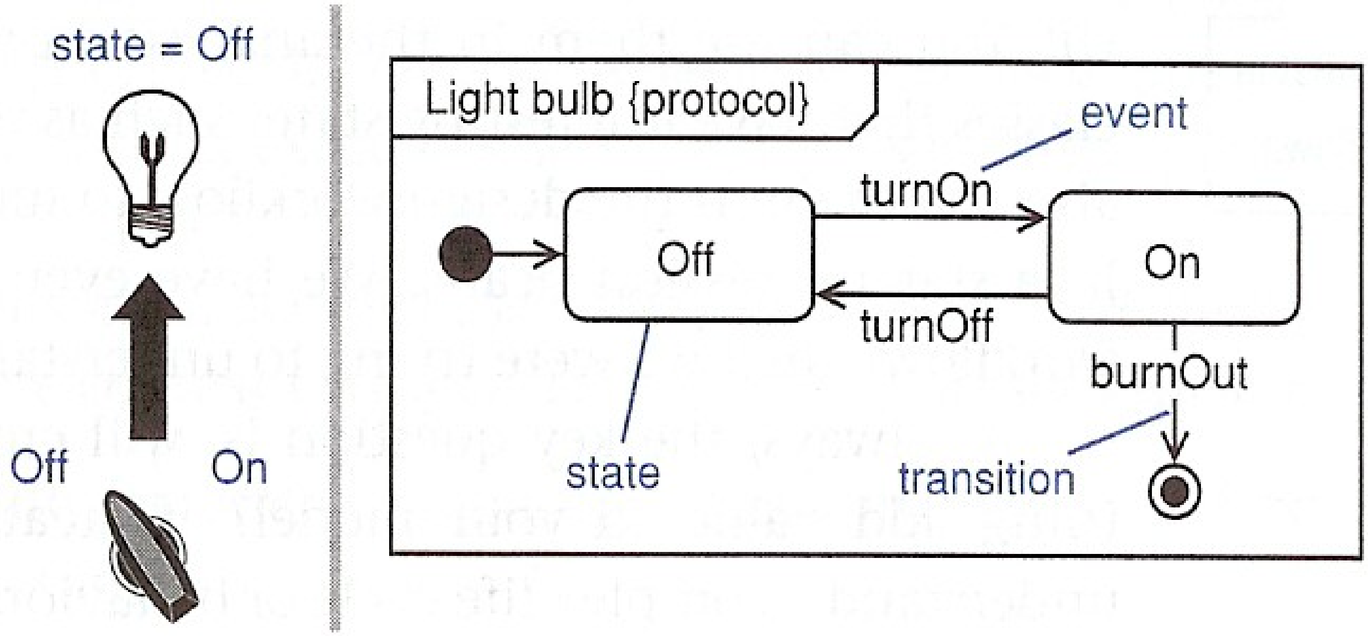
- Both activity diagrams and state machine diagrams model system behavior
- However, they have different semantics:
  - Activity diagrams are based on Petri Nets and usually model processes when several objects participate
  - State machines are based on Harel's statecharts and typically used to model single reactive objects

Introduction

- Reactive objects:
  - Respond to external events
  - May generate and respond to internal events
  - Have a lifecycle modeled as a progression of states, transitions and events
  - May have current behavior that depends on past behavior
- State machines are used to model behavior of items such as classes, use cases, subsystems, systems

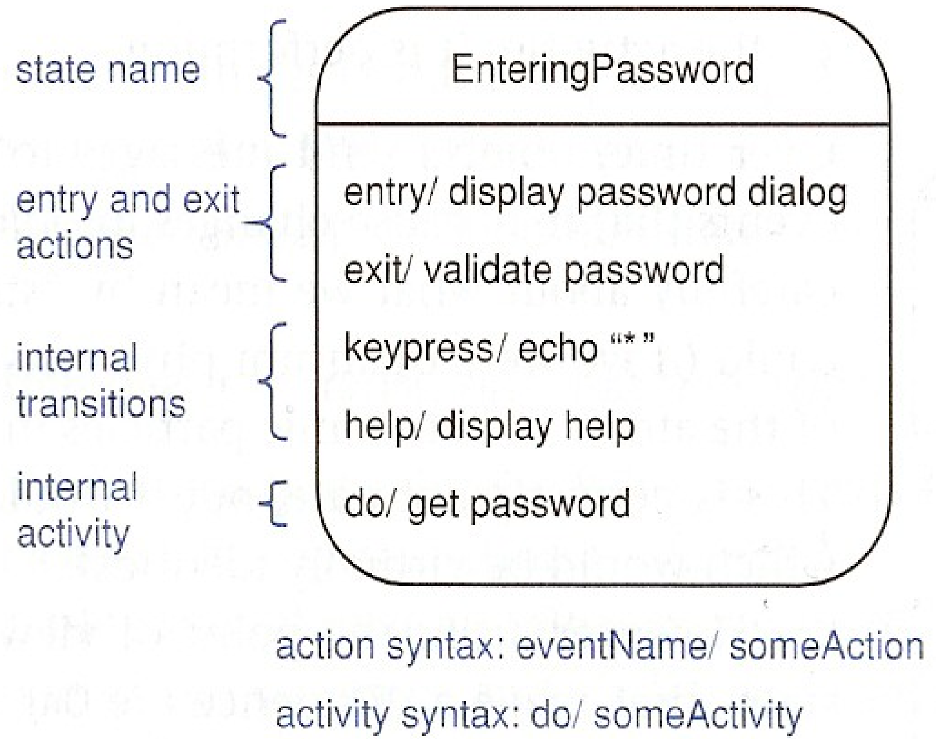
State machine diagrams

- There are three main modeling elements in state diagrams: *states, transitions*, and *events*.
- Example of a simple state machine, Fig. 21.2 [Arlow & Neustadt]



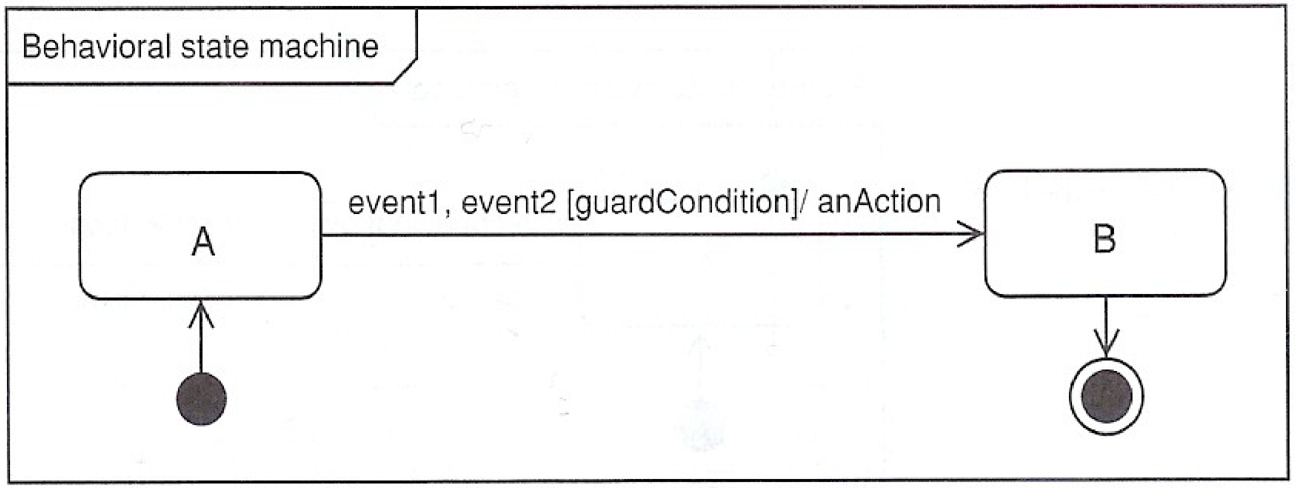
States

Summary of UML state syntax



Transitions

Summary of UML syntax for transitions in *behavioral state diagrams*, Fig.21.5

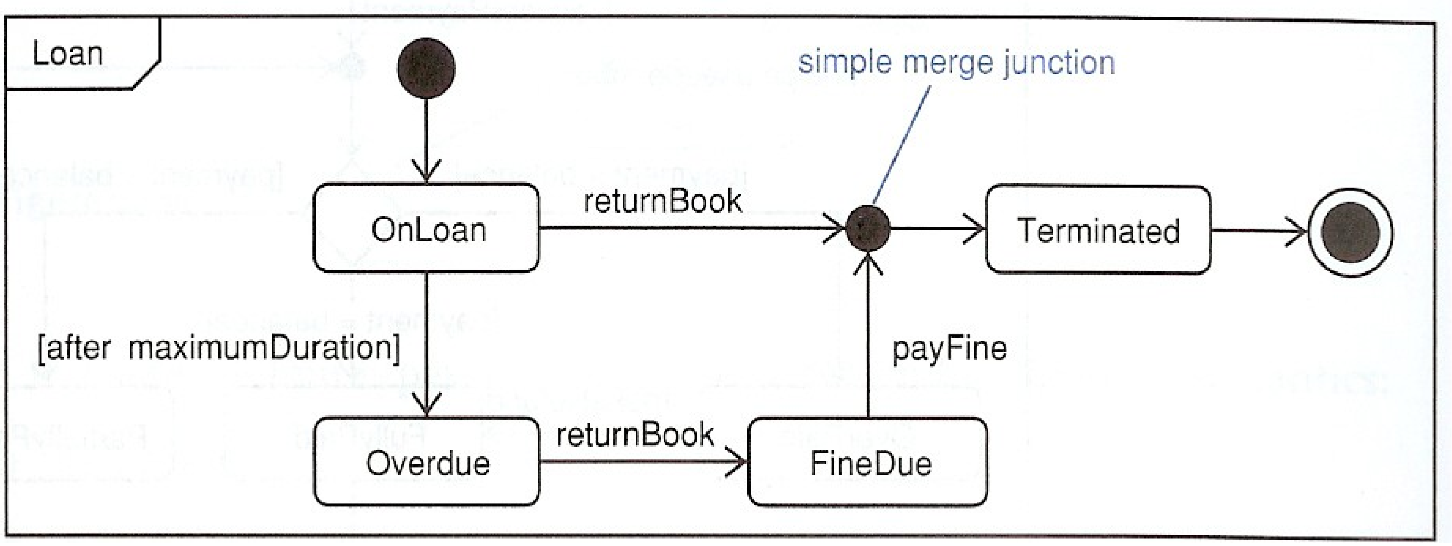


Where:

- event(s)= internal or external occurrence(s) that trigger the transition
- guardCondition = boolean expression, when *true the* transition is allowed
- anAction = some operation that takes place when the transition fires

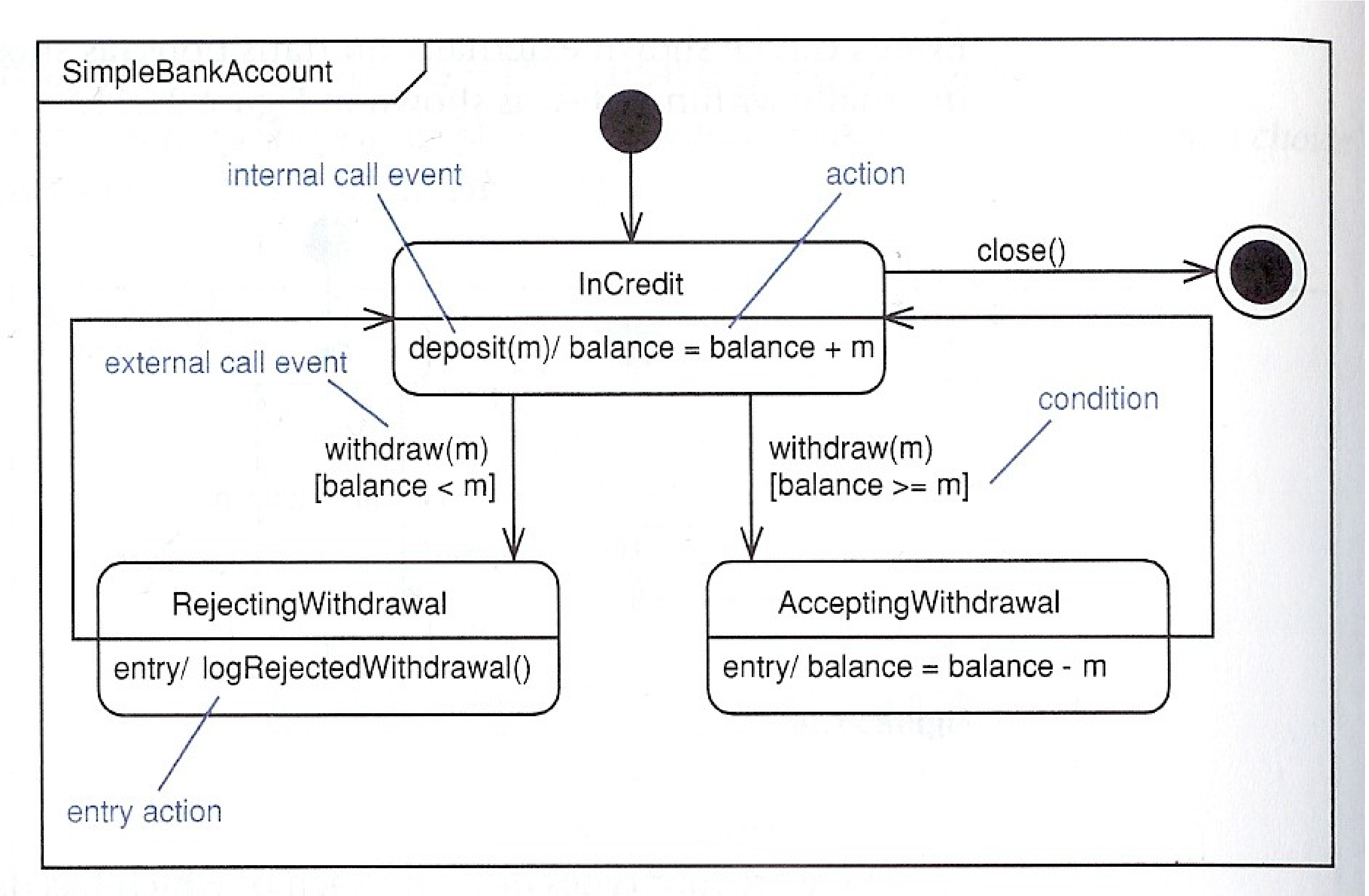
Transitions

A *junction pseudo-state* represents a point where
 transitions
merge or branch



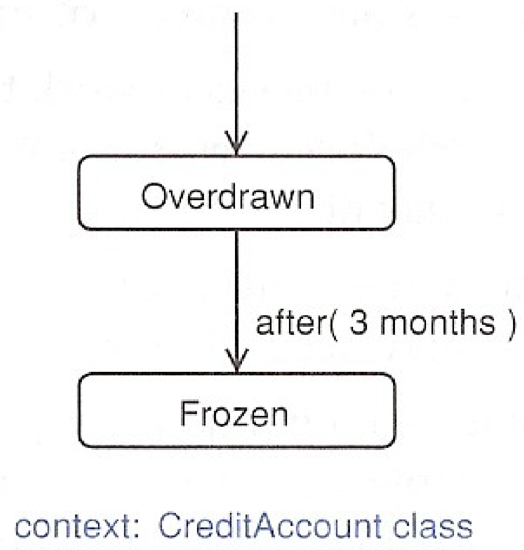
Events

Example of a call event, Fig.21.11 [Arlow & Neustadt 2005]



Events

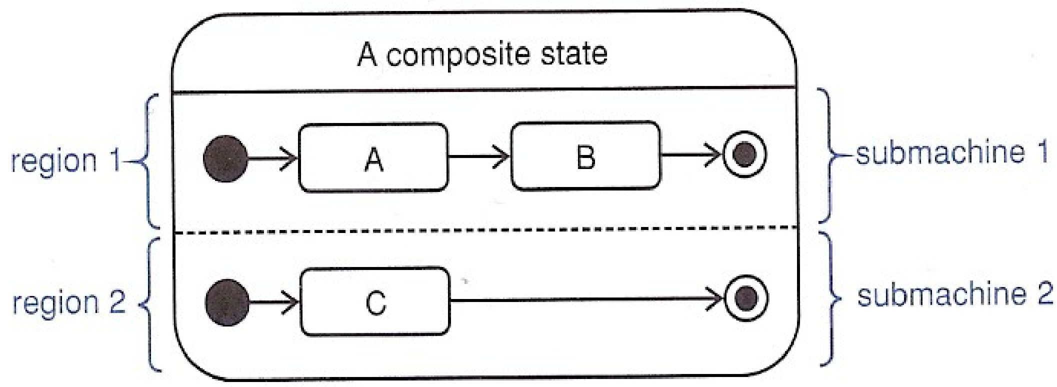
Time events are indicated by the keywords *when* and *after*.
Example of a time event:



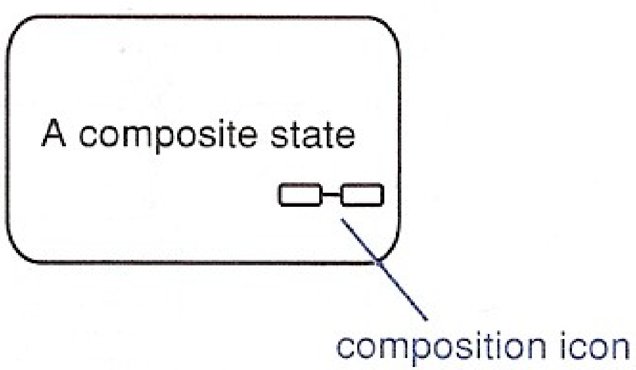
Example of a
state machine
[Dascalu 2001]

Composite states

- A *composite state* contains one or more nested state machines (*submachines*), each existing in its own *region*, Fig 22.2

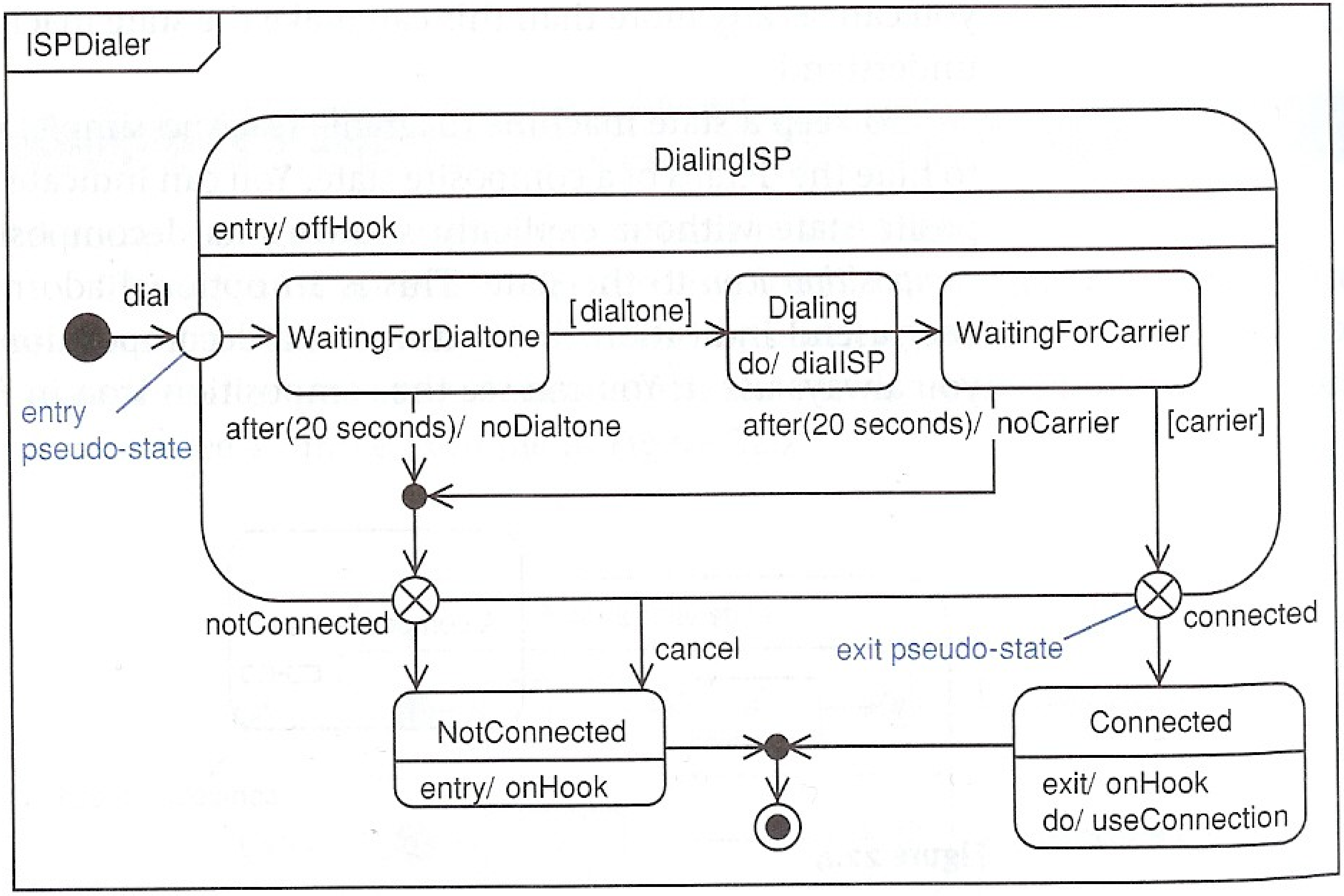


- The composition icon is shown in Fig. 22.4



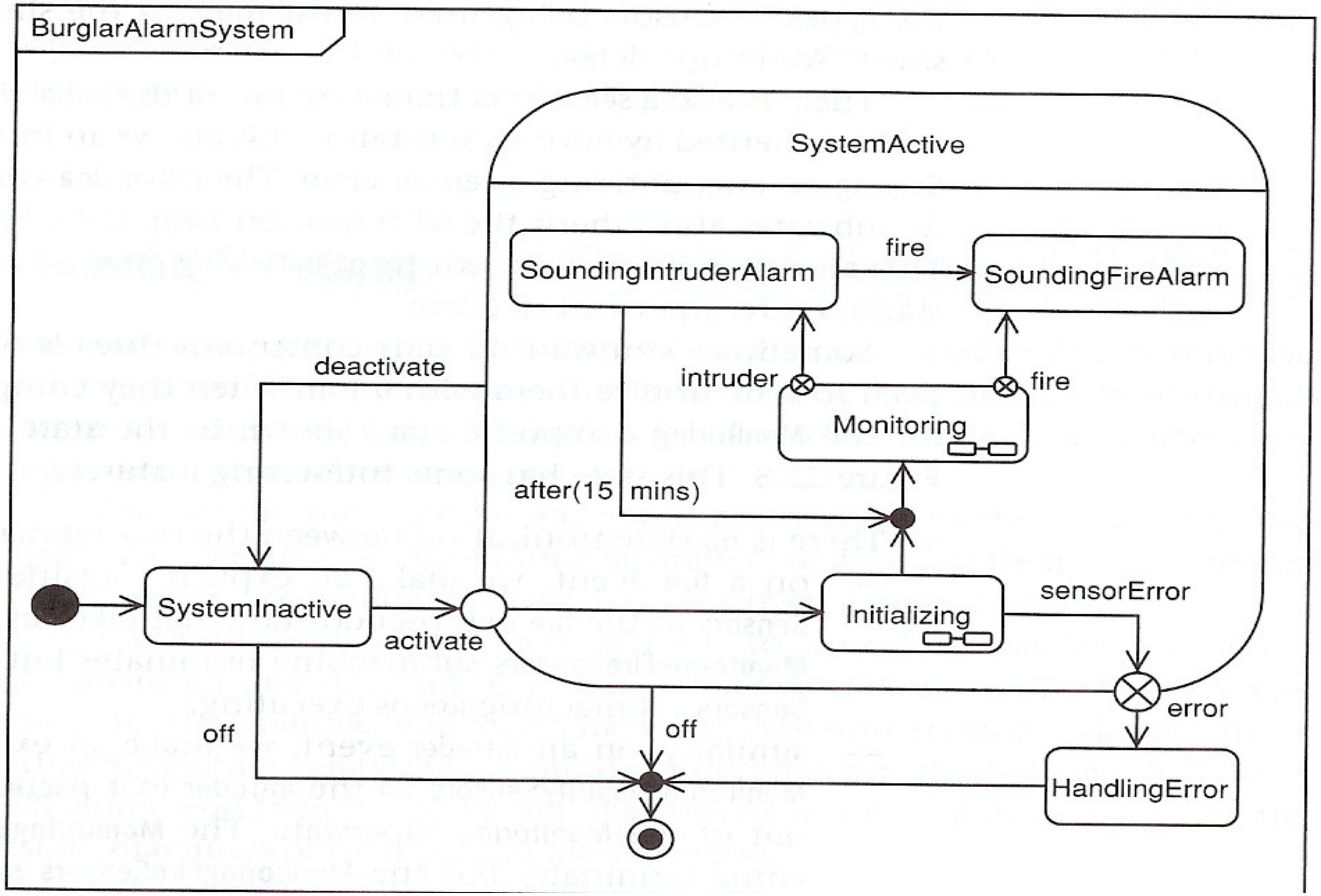
Simple composite states

- A superstate that contains a single region is called a *simple composite state*



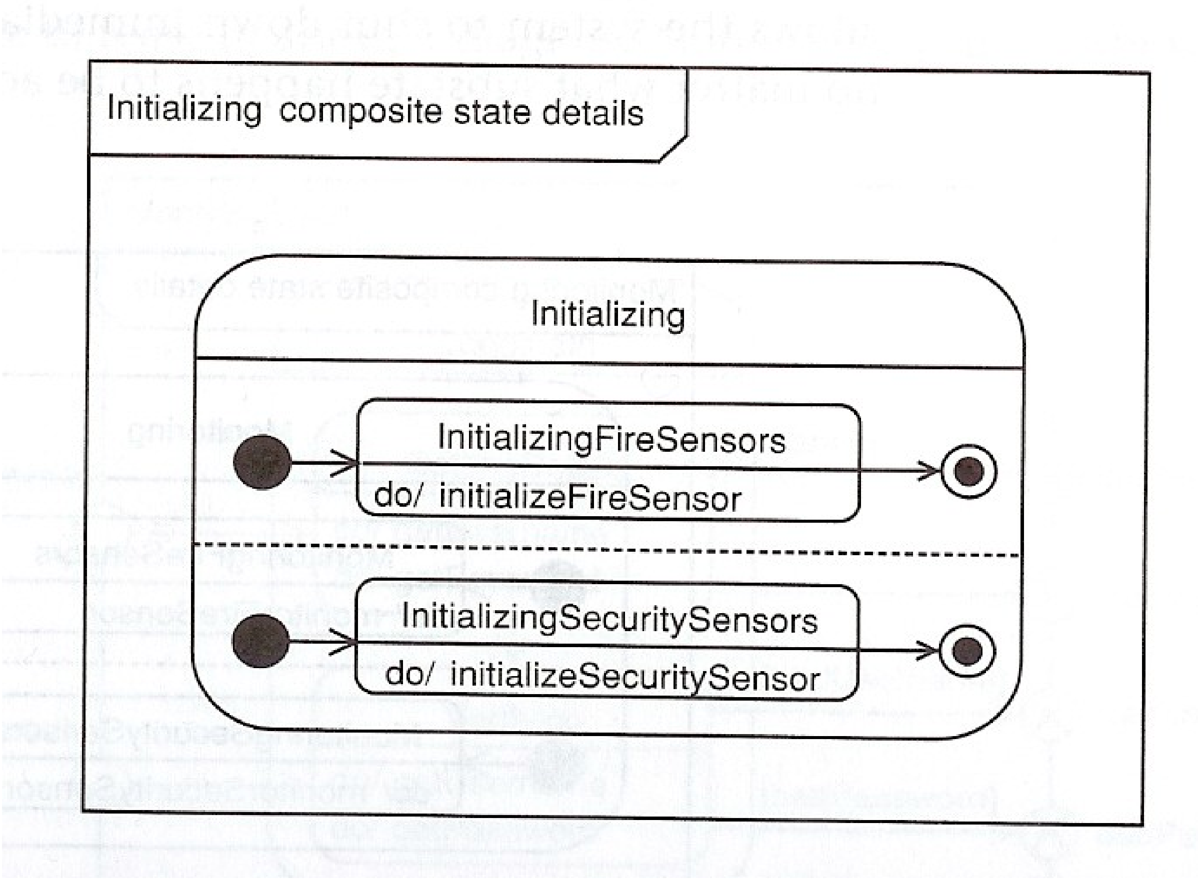
Orthogonal composite states

- Orthogonal composite states consist of two or more sub-machines that execute in parallel. Composite states below are *Initializing* and *Monit*



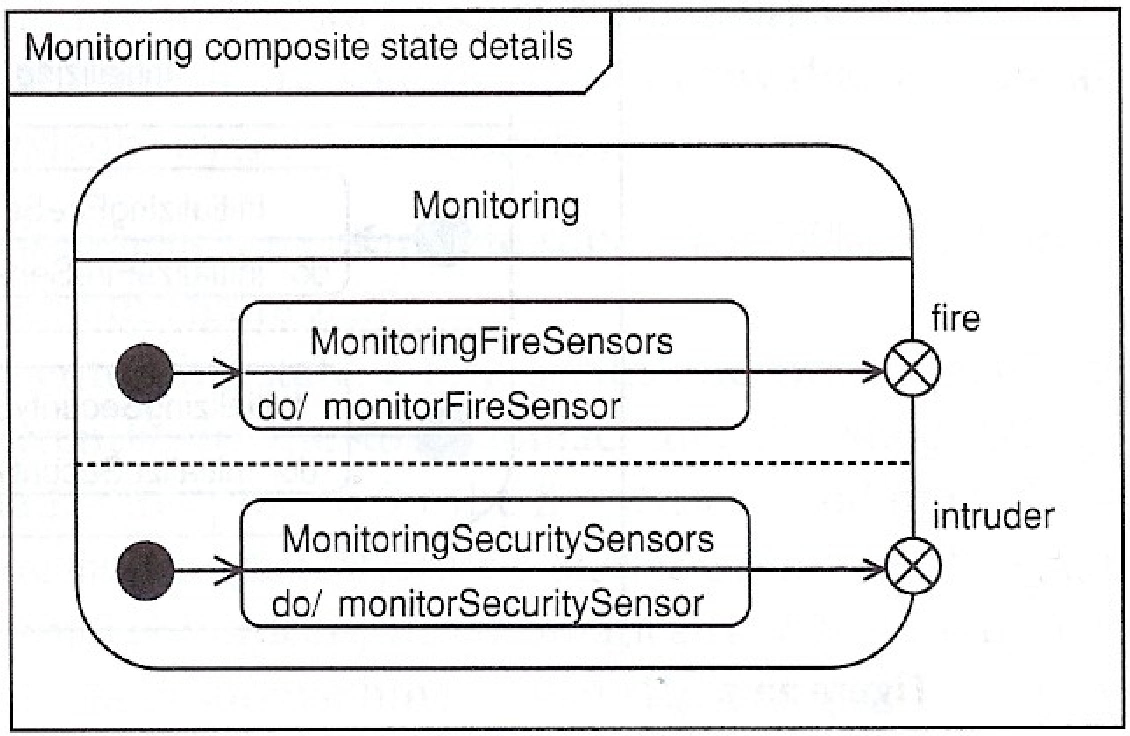
Orthogonal composite states

The composite state Initializing:



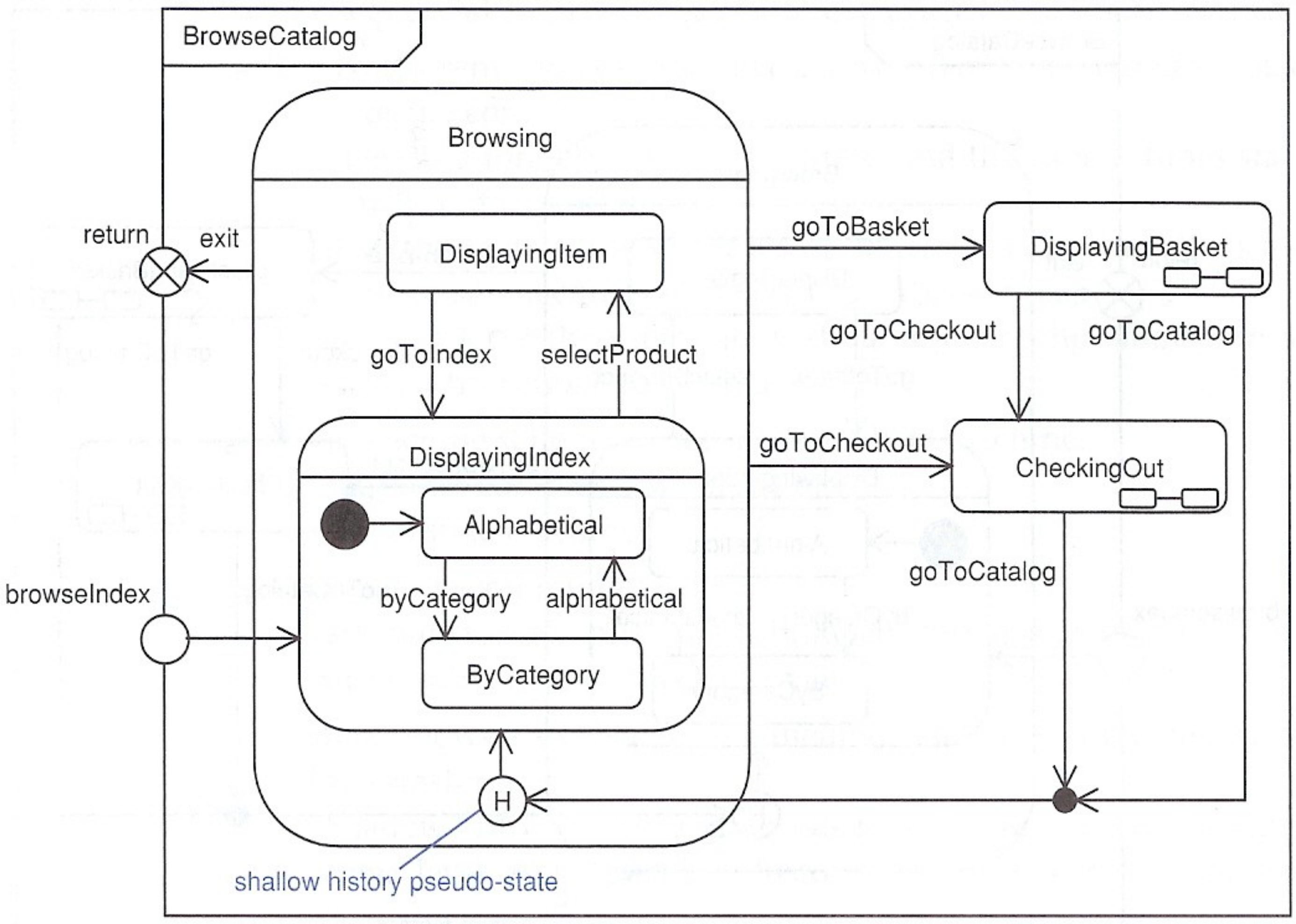
Orthogonal composite states

The composite state Monitoring



History

Example of using the *shallow history* indicator



History

Example of using the *deep history* indicator