AMSS Lecture 7: UML State Diagrams

Traian-Florin Șerbănuță

2025

Agenda

Session 1: Foundations of State Diagrams

What Are State Diagrams?

Definition

A UML State Machine Diagram models the *lifecycle* of an object — the states it goes through and how it transitions between them.

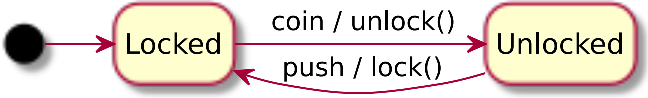
Use cases

- Modeling reactive systems.
- Describing event-driven behavior.
- Understanding object lifecycle and valid transitions.

Key Elements

- **States** (simple, composite)
- **Transitions** (with optional triggers, guards, and actions)
- **Initial/Final states**

Components of Transitions



Syntax

`trigger [guard] / action`

Trigger	event that initiates the transition (e.g., `coin`, `push`)
Guard	condition that must be true for transition to occur (e.g., `[balance > 0]`)
Activity / Action	operation executed when the transition occurs (e.g., `unlock()`, `displayMessage()`)

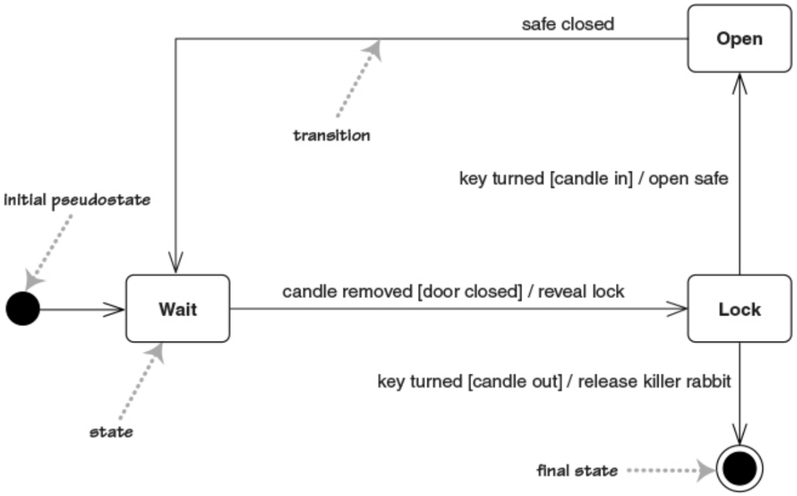
Notes on transitions

Syntax

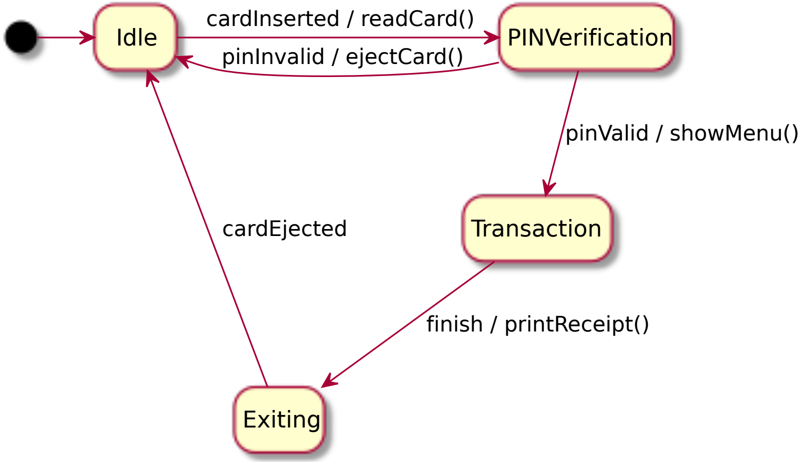
`trigger [guard] / action`

- All components of a transition are optional
  - no activity means no action is taken during transition
  - no guard means the transition is always taken when event occurs
  - no trigger is rarer (e.g., for activity states)
- You can take only one transition out of a state
  - transitions with same trigger must have mutually exclusive guards
- An event with no valid transition is ignored

Example: Gothic Castle Safe


safe closed
Open
transition
key turned [candle in] / open safe
initial pseudostate
Wait
candle removed [door closed] / reveal lock
Lock
key turned [candle out] / release killer rabbit
state
final state

Example: ATM Session Lifecycle



$$\begin{tikzcd}
\bullet \arrow[r] & \text{Idle} \arrow[r, bend left, "\text{cardInserted / readCard()}"] & \text{PINVerification} \arrow[l, bend left, "\text{pinInvalid / ejectCard()}"] \arrow[d, "\text{pinValid / showMenu()}"] \\
& \text{Exiting} \arrow[u, "\text{cardEjected}"] & \text{Transaction} \arrow[l, "\text{finish / printReceipt()}"]
\end{tikzcd}$$

Interactive Exercise: Identify Missing Transitions

Scenario: ATM State Diagram

- What happens if the card reader fails?
- How do we represent timeout conditions?

Task: Add error or timeout transitions to the diagram.

Recap

- **States** capture modes of behavior.
- **Transitions** define possible responses to events.
- **Actions** are side effects of transitions.
- State diagrams complement sequence/activity diagrams by focusing on *object lifecycles*.

Session 2: Advanced Modeling

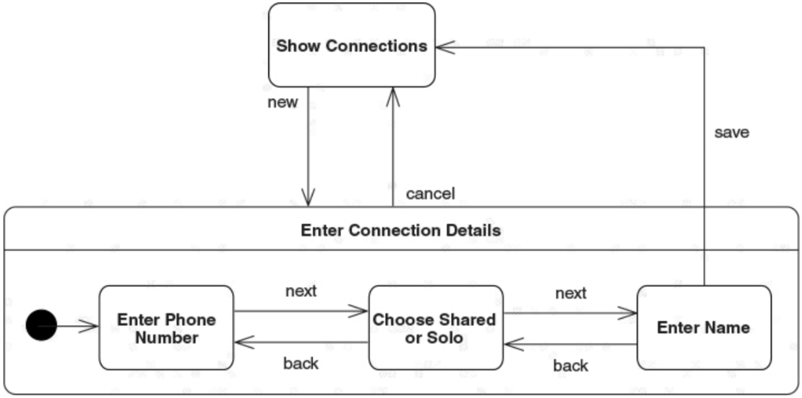
Superstates (Composite States) and Substates

Problem

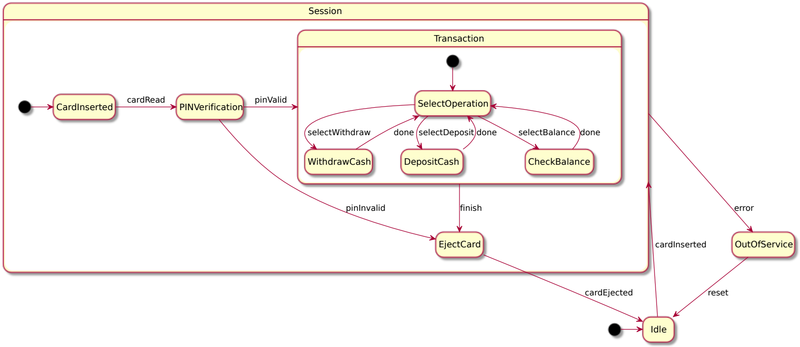
If several states share transitions / internal activities

Solution

Group them in a superstate and move shared behavior to it

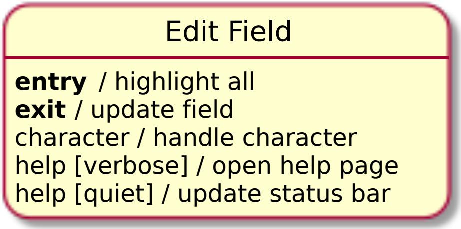


Example: ATM State Diagram with Composite States



Internal Activities

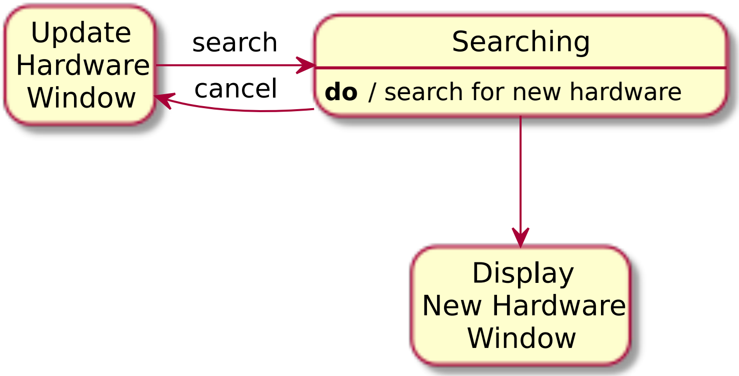
- React to events without changing state
- Similar to a self-transition
  - instead, we put the `trigger[guard]/action` within the state


Edit Field
entry / highlight all
exit / update field
character / handle character
help [verbose] / open help page
help [quiet] / update status bar

- Special entry/exit activities
  - Executed when entering / leaving the state
  - internal activities do not trigger them

Activity States (do-activities)

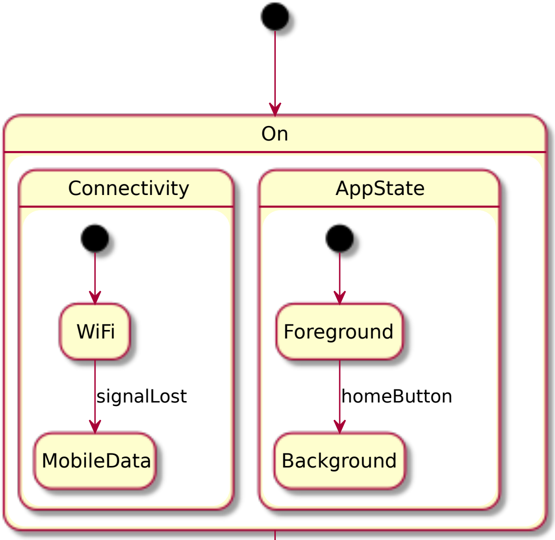
- States in which the object is doing some ongoing work
- Once ongoing activity is completed, a transition with no event is taken
- If an event occurs before the ongoing activity completes, activity is stopped



Concurrent (Orthogonal) States

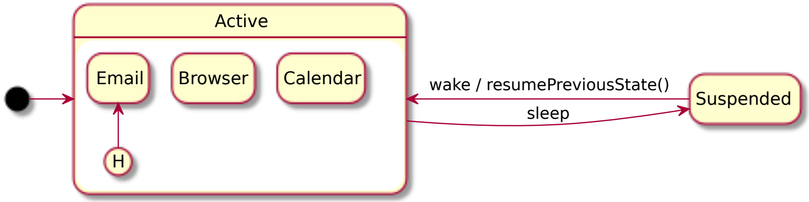
Some systems have *independent* aspects of behavior.

Example: Smartphone with concurrent regions:



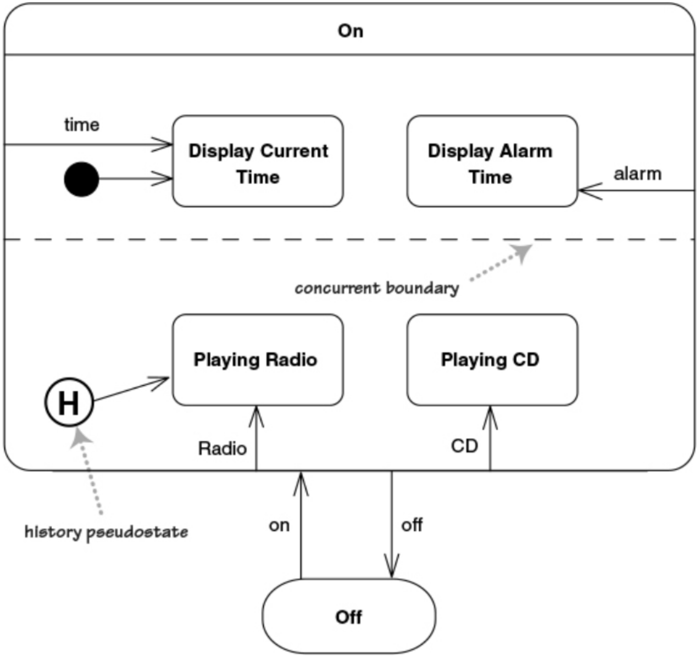
History Pseudo-States

Used to remember the **last active substate** when re-entering a composite state.



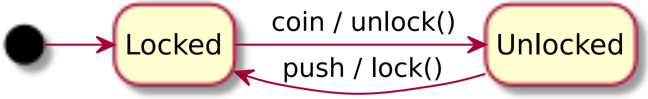
- In the diagram the history state points to the default state.
- Deep history (`H*`) can be used to remember nested hierarchy of substates

Example: Alarm clock with radio

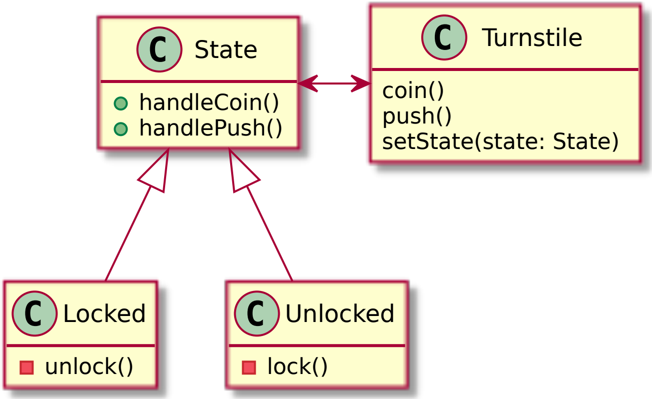


Session 3: Code Mapping

Remember the Turnstile State Diagram



Using the State Design Pattern



From UML to Code (Java Example)

Simple **State Pattern** implementation mapping UML concepts to code:

Source file

```java
class Turnstile {
    private State state = new Locked(this);
    void setState(State s) { state = s; }
    void coin() { state.handleCoin(); }
    void push() { state.handlePush(); }
}

class State {
    Turnstile turnstile;
    State(Turnstile turnstile) {
        this.turnstile = turnstile;
    }
    public void handleCoin() {
        printState();
```

Interactive Task

Description

A smart home system has two major features –**Security System** and **Climate Control**– operating *concurrently* when powered on. The system starts in an **Off** state and transitions to **Operational** when powered on.

Requirements

- When *entering* Operational, the system initializes.
- While Running, two subsystems work in parallel:
  - SecuritySystem
    - Must remember its previous substate when re-entered.
  - ClimateControl
    - Heats when temperature is below min threshold
    - Cools when temperature is above max threshold
    - Returns to Idle once temperature normalizes.
- The user can power **Off** the system at anytime

Wrap-Up

Concept	Description	Example
Simple state	Mode of behavior	Locked/Unlocked
Composite state	Grouped states	Playing (Buffering, Streaming)
Internal activities	transitions within same state	entry/exit
Activity state	States performing work	do/ search
Concurrent state	Independent regions	Connectivity, AppState
History	Remembers previous substate	Resume after pause

Takeaway: UML state diagrams help capture dynamic, event-driven aspects of systems and bridge toward implementable designs.