State Machine Diagram

Lecture B3 — Dynamic Modeling: Object Lifecycle

Prof. Ing. Lelio Campanile

Dipartimento di Matematica e Fisica
Università degli Studi della Campania Luigi Vanvitelli
Corso di Laurea Magistrale in Data Science

A.A. 2025/2026

Based on: Seidl et al. — *UML @ Classroom*, Ch. 7 (Springer, 2015)

Module B so far

- ✔ B1 **Sequence Diagram**
  Who sends what, in what order
- ✔ B2 **Activity Diagram**
  Flow of a process or algorithm
- ✔ B3 **State Machine Diagram** ←
  Lifecycle of a single object

What State Machines model

Sequence and Activity Diagrams model *processes*. A State Machine models the **lifecycle** of a *single object*: how it moves between **states** in response to **events**.

Key question: *"In which state is this object right now, and what can happen to it?"*

Classic examples: a `TrafficLight` cycling through Red/Yellow/Green; an `Enrollment` going through Requested / Approved / Active / Completed / Cancelled.

Topics covered

Core elements

- States: simple, initial, final
- Transitions and labels (event / guard / action)
- Internal transitions
- Entry and exit actions

Advanced elements

- Composite states (nesting)
- Orthogonal regions (concurrency)
- History pseudostates
- Best practices and pitfalls
- Comparison with Activity Diagram

Running example: **Enrollment** object in the Student Administration System — the lifecycle of one enrollment record.

Core Elements

States, transitions, events, guards, actions

What is a State Machine Diagram?

Definition

A **State Machine Diagram** (also called *statechart*) describes the **possible states** of an object and the **transitions** between them triggered by events.

It models the *reactive behavior* of a single object over its entire lifetime — from creation to destruction.

Key concepts

State	A situation in which the object waits for an event or satisfies a condition
Transition	A change from one state to another, triggered by an event
Event	An occurrence that can trigger a transition

When to use it

- Objects with complex lifecycle (orders, sessions, tickets, enrollments)
- Protocol specifications (network protocols, UI widgets)
- Embedded systems and controllers
- Any object where *what you can do depends on what state you're in*

States and Transitions

State

Drawn as a **rounded rectangle**. Contains the state name.

May also contain:

- `entry` / action executed on entering
- `do` / activity while in the state
- `exit` / action executed on leaving

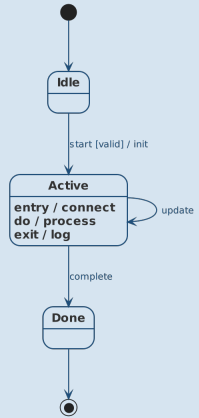
Initial pseudostate: filled black circle. **Final state**: bullseye.

Transition label syntax

```
event [guard] / action
```

All three parts are optional. The guard is a

Notation



Transition Labels — `event [guard] / action`

Event

Triggers the transition.

Types of events:

- *Signal event*: external signal received
- *Call event*: operation called
- *Time event*: `after(5s)`
- *Change event*: `when(x>0)`
- *Completion*: no label (implicit)

Guard

Boolean condition in `[brackets]`.

The transition fires **only if** the guard is true when the event occurs.

Example:
`submit [credits >= 30]`

If multiple transitions have the same event, guards must be **mutually exclusive**.

Action

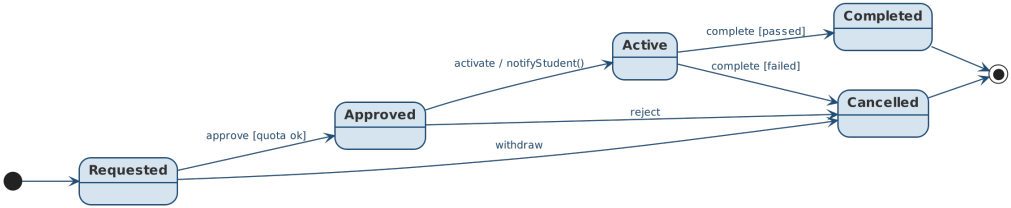
Executed **instantaneously** when the transition fires (after exit of source state, before entry of target state).

Example:
`submit / sendConfirmation()`

Internal transition: same source and target state; does *not* trigger entry/exit actions.

Example: Enrollment Lifecycle

Student Administration System — lifecycle of one `Enrollment` object



Advanced Elements

Composite states, orthogonal regions, history

Composite States

Definition

A **composite state** contains one or more *nested* state machines (regions).

Why use them:

- Factor out common transitions: a transition from the composite state applies to *all* sub-states
- Reduces diagram size significantly
- Models hierarchical behavior naturally

A transition entering a composite state enters its **initial pseudostate** (unless targeting a specific sub-state directly).

Example: Active is composite

Orthogonal Regions — Concurrency within a State

Definition

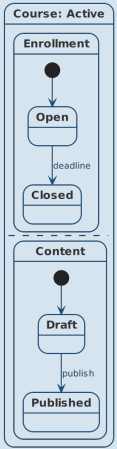
A composite state can be divided into **orthogonal regions** separated by a dashed line. Each region has its own sub-state machine and runs **concurrently**.

The object is *simultaneously* in one sub-state per region.

The composite state is exited when **all** regions reach their final sub-state.

Useful for: an object with two independent aspects that evolve in parallel, e.g., a document that can be *edited* and *reviewed* simultaneously.

Example: Course status



History Pseudostate

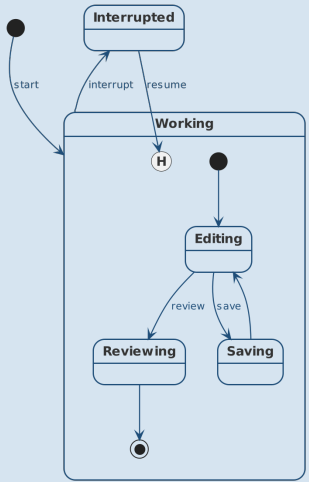
Definition

The **history pseudostate** (H inside a small circle) allows a composite state to **remember** the last active sub-state when it was exited.

When re-entering the composite state via the history pseudostate, execution resumes from the remembered sub-state rather than from the initial pseudostate.

Shallow history (H): remembers only the direct sub-state.
Deep history (H*): remembers the full nested configuration.

Example: interrupted session



State Machine vs. Activity Diagram

Choosing the right tool

State Machine vs. Activity Diagram

	State Machine Diagram	Activity Diagram
Models	Lifecycle of *one object*	Flow of a *process or algorithm*
Driven by	*Events* (external triggers)	*Completion* of actions
Focus	Object *state* at each moment	*Sequence* of steps
Participants	One object	Multiple actors / systems
Guards	On *transitions*	On *decision* outgoing edges
Use when...	Object has state-dependent behavior	Process has branching / parallelism

Rule of thumb: if you find yourself asking *"what state is this object in?"* — use a State Machine. If you ask *"what step comes next?"* — use an Activity Diagram.

Best Practices and Pitfalls

Best practices

- **One diagram per class**: model the lifecycle of a single class; do not mix multiple objects in one diagram
- **Name states as nouns or adjectives**: `Active`, `Approved`, not `doActivate`
- **Ensure all transitions are reachable**: no dead states
- **Use composite states** to factor out common cancellation or error transitions
- **Keep it readable**: 5–8 states maximum in a non-composite diagram

Common pitfalls

- **Confusing state with activity**: a state is a *condition*, not a step being executed
- **Missing guards on exclusive branches**: if two transitions share an event, guards must be mutually exclusive and complete
- **Forgetting entry/exit actions**: they execute on *every* entry/exit, not just some
- **Using State Machine for process flows**: if multiple objects are involved, prefer Sequence or Activity Diagram

Notation Elements

Element	Notation	Description
Initial pseudostate	Filled circle	Starting point; not a state
State	Rounded rectangle	Object is waiting / satisfying a condition
Final state	Bullseye	Object lifecycle ends
Transition	Arrow + label	`event [guard] / action`
Internal transition	Label inside state	Fires without leaving the state
Composite state	State with nested SM	Groups sub-states with common transitions
Orthogonal regions	Dashed separator	Concurrent sub-state machines
History (H)	Circle with H	Resume from last sub-state on re-entry
Deep history (H*)	Circle with H*	Resume full nested configuration

Key points

State Machines model reactive objects

They describe how a *single object* responds to events over its lifetime. The behavior depends entirely on the *current state*.

Transitions: event / guard / action

An event triggers a transition. The guard (optional) controls when. The action (optional) executes instantaneously as the transition fires.

Composite states reduce complexity

Nesting states avoids duplicating common transitions. A single *cancel* transition from the composite state applies to all sub-states at once.

Choose the right diagram

State Machine: one object, event-driven lifecycle.
Activity: process flow, multiple actors.
Sequence: who sends what to whom.

Module B — Behavioral Modeling: Wrap-up

Three complementary views

- ✔ B1 **Sequence Diagram**
  Object interactions, one UC scenario
- ✔ B2 **Activity Diagram**
  Process flow, forks, swimlanes
- ✔ B3 **State Machine Diagram**
  Object lifecycle, event-driven

All grounded in Module A

Every behavioral diagram references the *classes* and *use cases* from Module A. The Student Administration System was modeled from five complementary perspectives

UML in the software process

Requirements	Use Case Diagram
Analysis	Class (overview)
Design	Class (full), Sequence, Activity, State Machine
Impl.	Code generation from Level 3 Class Diagrams

Modeling tools

draw.io (web, free)
PlantUML (text-based, git-friendly)
StarUML (desktop, UML 2.x)
Mermaid (Markdown-embedded diagrams)

Thank you

Prof. Ing. Lelio Campanile

lelio.campanile@unicampania.it

Dipartimento di Matematica e Fisica
Università degli Studi della Campania Luigi Vanvitelli

Elements of Software Engineering and Information Systems

Corso di Laurea Magistrale in Data Science