Sequence Diagram

Lecture B1 — Dynamic Modeling: Object Interactions

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. 4 (Springer, 2015)

Where we are

In Module A we modeled the *structure* of the Student Administration System (Use Case and Class Diagram). We answered: *what exists and how it is organized*.

Now we ask: **what happens at runtime, and in what order?**

Module A (static)

- ✔ Use Case Diagram
- ✔ Class Diagram

What entities exist?
How are they related?

Module B (dynamic)

- ✔ **Sequence Diagram** ←
- ✔ Activity Diagram
- ✔ State Machine

What happens?
In what order?

Key link

Each Sequence Diagram corresponds to **one scenario** of a Use Case from Module A.

The objects that interact correspond to **classes** from the Class Diagram.

Topics covered

Core elements

- Interaction partners (lifelines)
- Message types: synchronous, asynchronous, return
- Activation bars
- Object creation and destruction
- Found and lost messages

Combined fragments

- `alt` — alternatives (if/else)
- `opt` — optional (if)
- `loop` — repetition
- `par` — parallel execution
- `ref` — reference to another diagram

Running example: **"Enroll in course"** scenario of the Student Administration System.

Core Elements

Lifelines, messages, and activation bars

What is a Sequence Diagram?

Definition

A **Sequence Diagram** describes the interaction between *interaction partners* for a specific **scenario**. It shows which messages are exchanged, in which **order**, and on which **lifeline** each action takes place.

Key properties

- Models **one scenario** (one execution path) of a use case
- Time flows **top to bottom**
- Objects are shown as vertical **lifelines**
- Interactions are shown as **horizontal arrows**
- Covers *syntax*, *semantics* and when to use it

When to use it

- To specify the detailed flow of a UC scenario
- To design method calls and API contracts
- To document communication protocols
- To read existing systems (e.g., OAuth 2.0 flow, REST API call sequence)

Interaction Partners — Lifelines

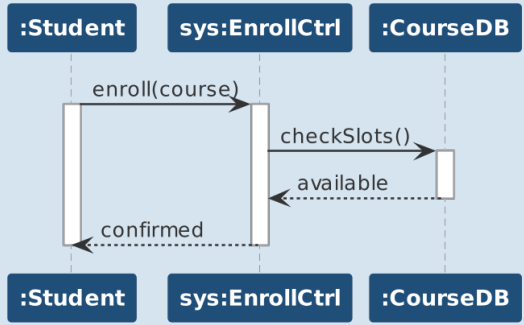
Definition

Interaction partners are the participants in a sequence diagram. Each is shown as a **lifeline**:

- A *head box* at the top identifying the object and its class: `object : Class`
- A vertical *dashed line* below representing the passage of time
- An *activation bar* (thin rectangle) on the lifeline when the object is active

The head box may show `: ClassName` (anonymous object) or `name : ClassName` (named object).

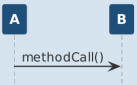
Lifeline notation



Message Types

Synchronous message

Sender **waits** for receiver to complete. Filled arrowhead. Typical: method calls.



Asynchronous message

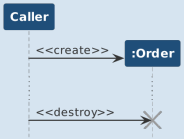
Sender **does not wait**. Open arrowhead. Typical: events, signals, callbacks.

Return message

Shows the **return** from a synchronous call. Dashed open arrowhead. Often omitted when obvious.

Object creation / destruction

«create» points to the head box of a new lifeline. «destroy» precedes an **X** mark at the end of the lifeline.



Found and Lost Messages

Found message

A message whose **sender is unknown** or outside the scope of the diagram. Shown with a *filled circle* at the start of the arrow.

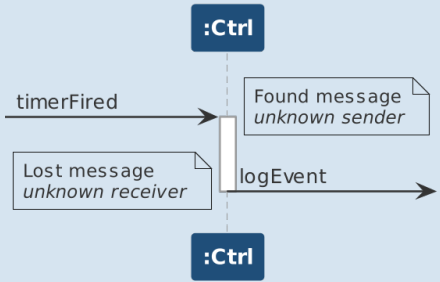
Typical: initial trigger from the environment, interrupt, timer event.

Lost message

A message whose **receiver is unknown** or outside the diagram scope. Shown with a *filled circle* at the end of the arrow.

Typical: message sent to an external system not modeled in this diagram.

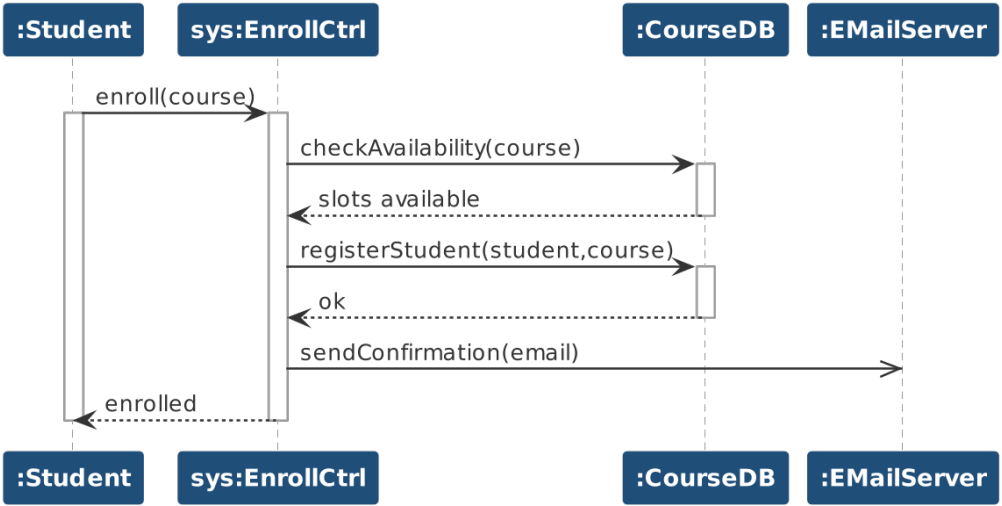
Notation — Found & Lost



Found: unknown sender — filled circle at the *source*.
Lost: unknown receiver — filled circle at the *destination*.

Example: “Enroll in course” — standard scenario

Student Administration System — Use Case A2, standard process



Combined Fragments

Controlling flow: alt, opt, loop, par

Combined Fragments — Concept

Definition

A **combined fragment** is a labeled region of a sequence diagram that controls **conditional** and **repetitive** behavior. The operator keyword appears in the **top-left corner** of the fragment box.

Control-flow operators

`alt`	if / else if / else
`opt`	optional (if only)
`loop`	repetition
`break`	exit enclosing fragment

Concurrency and reuse operators

`par`	parallel execution
`ref`	reference to another diagram
`seq`	weak sequencing (default)
`critical`	atomic region

In this course we focus on: `alt`, `opt`, `loop`, `par`, `ref`. These cover the vast majority of practical modeling needs.

`alt` — Alternatives

Semantics

Exactly one of the operands is executed, based on the *guard condition* in brackets `[condition]`.

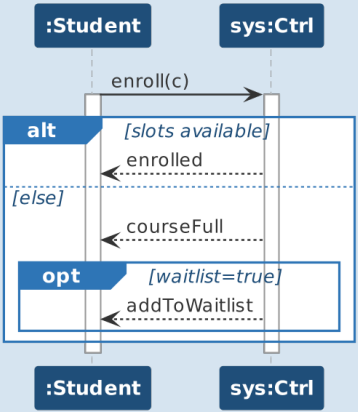
Operands are separated by a **dashed horizontal line**.

The `[else]` guard may be used for the default branch.

Equivalent to an **if/else if/else** construct.

Use `alt` when the scenario branches based on a condition — e.g., slot available vs. not available.

Example: enroll with `alt`

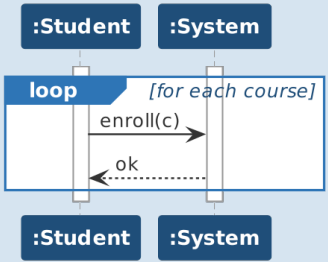


loop and par fragments

loop — Repetition

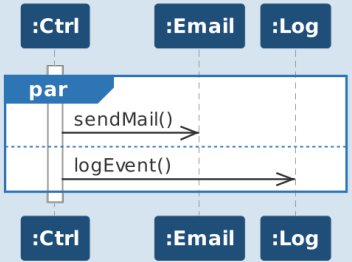
The enclosed interaction is executed **repeatedly**.

Guard syntax: `[min, max]` or `[condition]`.



par — Parallel execution

The enclosed operands are executed **concurrently**. Operands separated by dashed lines. Order within each operand is preserved; across operands is interleaved.



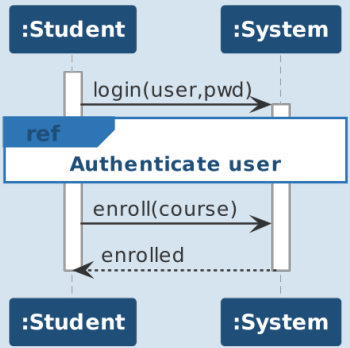
`ref` — Diagram Reference

Semantics

The `ref` fragment **references another sequence diagram** by name. It acts as a macro or sub-routine call.

- Keeps individual diagrams **manageable in size**
- Enables **reuse** of common sub-sequences (e.g., "Authenticate user" referenced from many diagrams)
- The referenced diagram *replaces* the `ref` box at execution

Example



Sequence Diagrams in Practice

Reading, writing, and common pitfalls

Sequence Diagrams and API Design

Key insight for Data Scientists

A Sequence Diagram read **from top to bottom** describes the *contract* between components. Each arrow crossing a lifeline boundary is a **method call or API endpoint**.

Reading a Sequence Diagram

- Each **head box** = one class or external system
- Each **arrow** = one method call or message
- **Activation bars** show when an object is processing
- **Dashed return arrows** show what is returned

Common pitfalls

- **Too many lifelines**: keep to 3–5 per diagram; use `ref` for sub-sequences
- **Missing return messages**: show returns for synchronous calls unless truly trivial
- **Implementation detail**: stay at design level; avoid micro-calls
- **Confusing with Activity Diagram**: Sequence focuses on *who* sends *what*; Activity on *flow of control*

Notation Elements

Element	Notation	Description
Lifeline	Rect. + dashed line	Interaction partner over time
Activation bar	Thin rectangle	Object is active/processing
Sync. message	Filled arrowhead	Sender waits for return
Return message	Dashed arrow	Return from sync. call
Async. message	Open arrowhead	Sender does not wait
«create»	Arrow to head box	Object instantiation
«destroy»	Arrow + X mark	Object destruction
Combined fragment	Labeled rectangle	`alt`, `opt`, `loop`, `par`, `ref`

Key points

Sequence Diagrams are dynamic

They model *one scenario* of a use case: a specific sequence of messages between objects over time. Time flows top to bottom.

One diagram per scenario

A Sequence Diagram models the *standard process* or one *alternative process* of a UC. Use `alt` to keep variants in one diagram or split into separate diagrams.

Combined fragments control flow

`alt`: branches (if/else)
`opt`: optional (if)
`loop`: repetition
`par`: concurrency
`ref`: reuse sub-sequences

Relationship to other diagrams

Each arrow maps to a **method** in the Class Diagram. The objects on lifelines are **instances of classes** from Module A.

What's next

Lecture B2 — Activity Diagram

Modeling the *flow of a process* or workflow.

Where Sequence Diagrams focus on *object interactions*, Activity Diagrams focus on *activities and control flow* — including forks, joins, decisions, and swimlanes.

Seidl et al., Ch. 5

Lecture B3 — State Machine Diagram

Modeling the *state-driven behavior* of a single object in response to events.

Particularly useful for objects with complex lifecycle (e.g., an `Order` that goes through states: *created* → *submitted* → *paid* → *shipped*).

Seidl et al., Ch. 7

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