Université de Paris

Génie Logiciel
UML to model the dynamic – part 2

Sylvain Lobry

18/11/2022

UML to model the structure

Before we start

- Notes: more than one model acceptable for the same system because:
  - Different interpretations
  - Different levels of details
- Syntax of the diagram: **extremely** important!
- UML is a visual language so the syntax and the style is essential to the comprehension

UML to model the dynamic

Types of diagrams

- UML defines 13 diagrams in 3 categories which can define a system according to different points of view
- Structure diagrams
  - Class Diagram, Object Diagram, Component Diagram, Composite Structure Diagram, Package Diagram and Deployment Diagram
- Behavior diagrams
  - Use Case Diagram, Activity Diagram and State Machine Diagram
- Interaction diagrams
  - Sequence Diagram, Communication Diagram, Timing Diagram and Interaction Overview Diagram

UML to model the dynamic

Types of diagrams

- UML defines 13 diagrams in 3 categories which can define a system according to different points of view
- Structure diagrams
  - Class Diagram, Object Diagram, Component Diagram, Composite Structure Diagram, Package Diagram and Deployment Diagram
- Behavior diagrams
  - Use Case Diagram, Activity Diagram and State Machine Diagram
- Interaction diagrams
  - Sequence Diagram, Communication Diagram, Timing Diagram and Interaction Overview Diagram

UML to model the dynamic

The dynamic

- We have seen a diagram allowing to model the interaction between internal (system) and external (actor) entities
- How do they interact?
- Sequence diagram: model the **temporal** aspects
- Communication diagram: model the **spatial** aspects

Sequence diagram

Sequence diagram

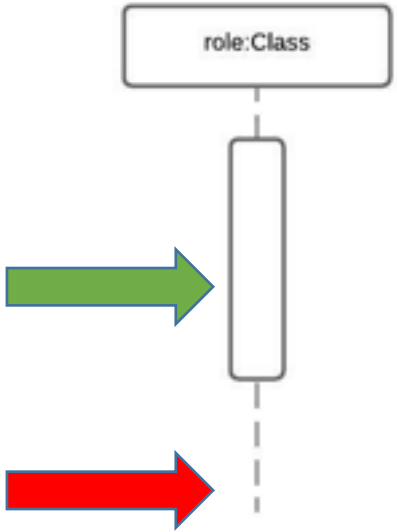
- A **sequence diagram** describes the interactions between different objects by showing which messages transmitted between them.
- Shows:
  - How do objects interact with each other
  - What information do they exchange (optional)
  - In what order do they communicate

- Use it to show how small methods are sequenced.
- One use case should come with one sequence diagram

Sequence diagram

Lifeline of an object

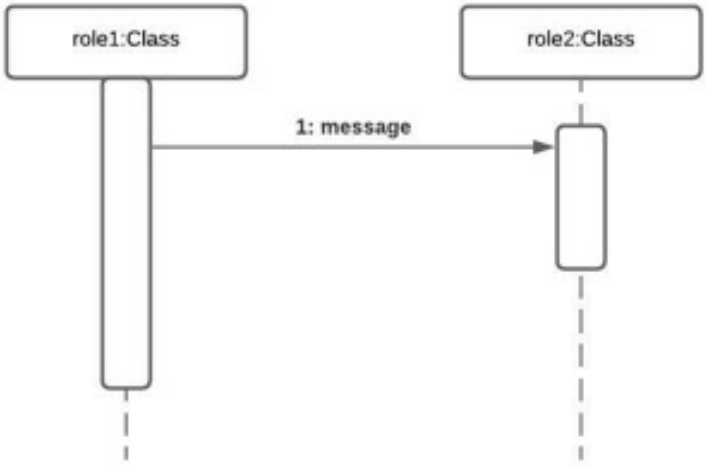
- Each instance of an object is named with the notation “role: Class”. If there is no ambiguity, “:Class” can be sufficient.
- Each instance of an object has a lifeline, shown with a dashed lined.
- The diagram is read top to bottom: time increases when we go down.
- Activation period represents the time during which the instance is active (i.e. runs a method)



Sequence diagram

Sending messages

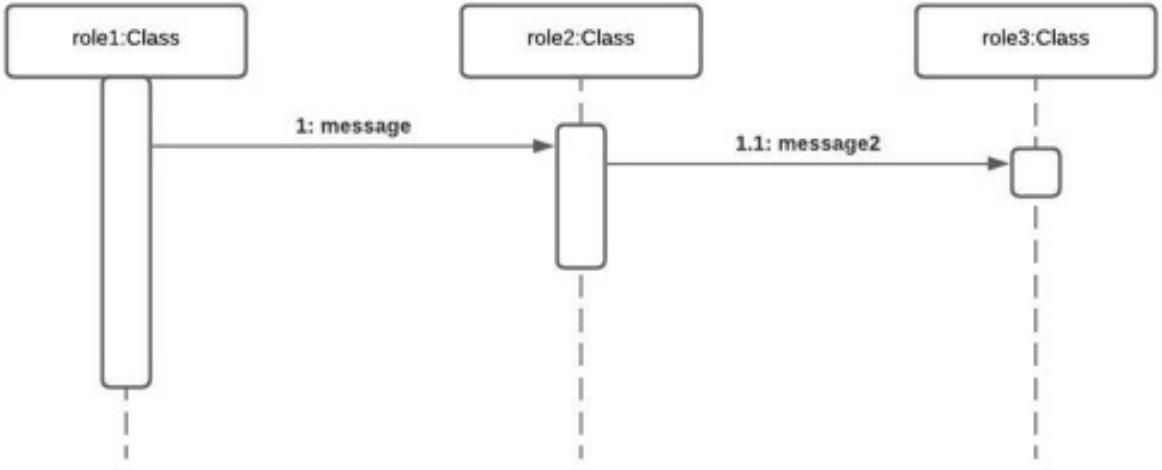
- Horizontal arrow from the sender's lifeline to the receiver.
- One message = 1 number (in sequential order) + a name.



Sequence diagram

Sending messages

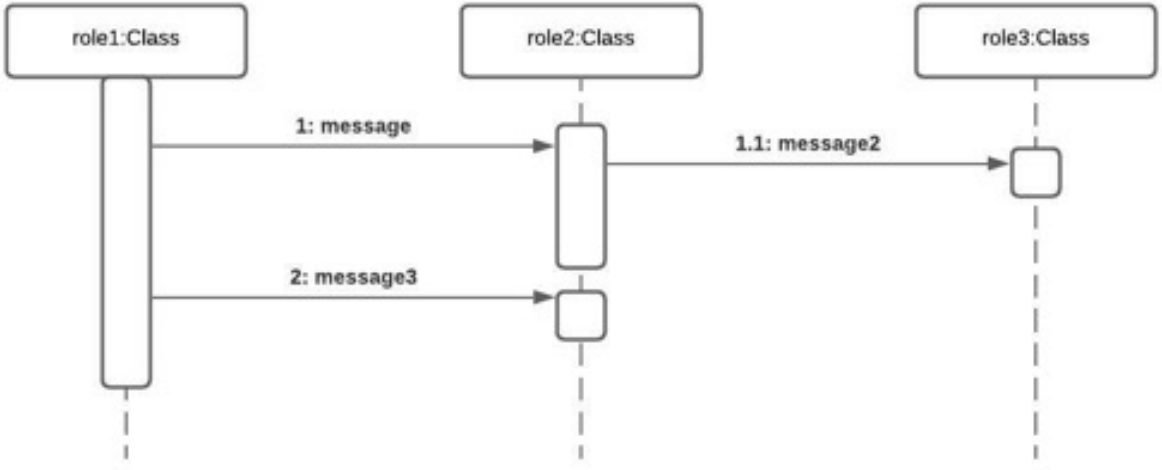
- Horizontal arrow from the sender's lifeline to the receiver.
- One message = 1 number (in sequential order) + a name.
- When a message is sent while the previous one is not finished: sub-numbering.



Sequence diagram

Sending messages

- Horizontal arrow from the sender's lifeline to the receiver.
- One message = 1 number (in sequential order) + a name.
- When a message is sent while the previous one is not finished: sub-numbering.



Sequence diagram

Sending messages

- The message can be
  - Synchronous: the sender stops its activity while the recipient is working on the message

  - Asynchronous: the sender does not stop its activity

  - Reply

- Possible to send message to itself

Sequence diagram

Sending messages

- The message does not have to have a name. In this case:

1: *

- The message can embed data through parameters

2: message(data)

3: message(param= data)

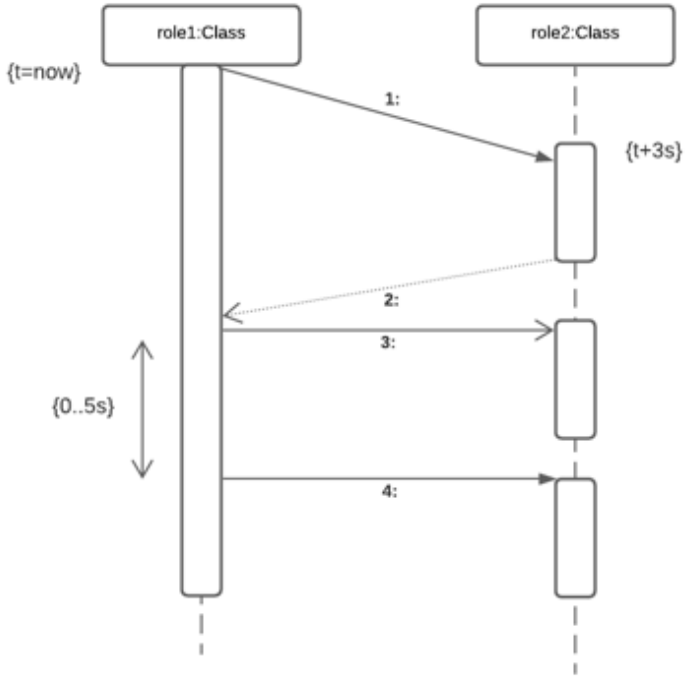
- Parameters can be omitted through ‘-’

4: message(-)

Sequence diagram

Taking time into account

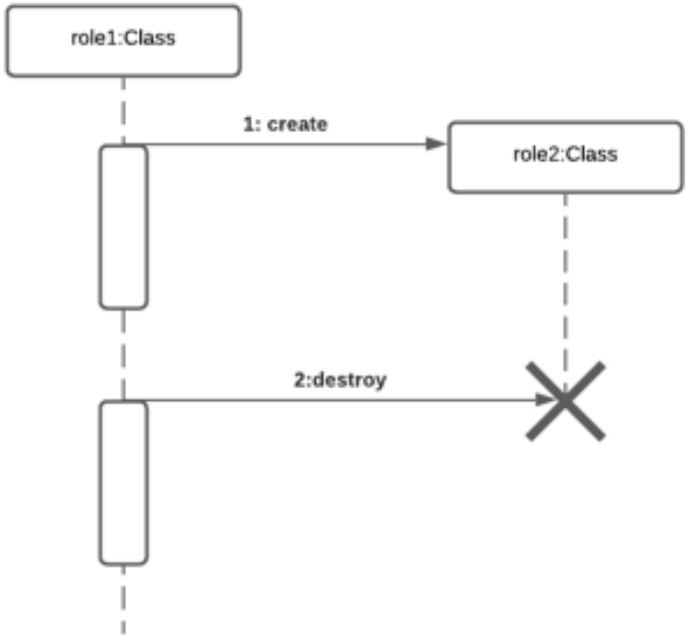
- If necessary, you can give time indication in your sequence diagram.
- Between brackets



Sequence diagram

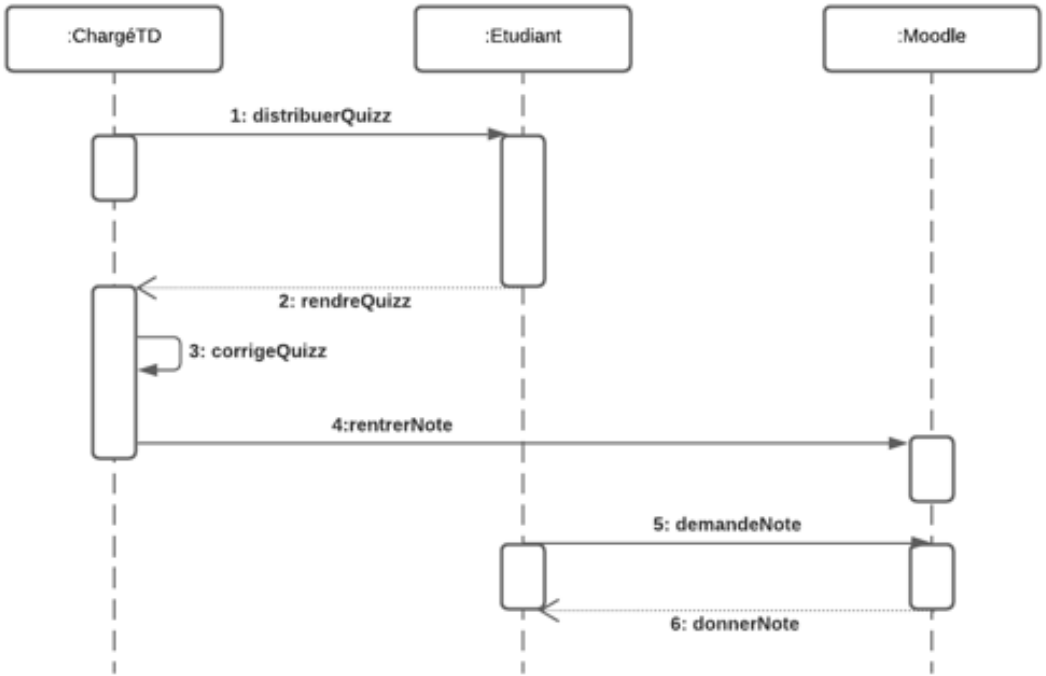
Creating and destructing object

- Possible to create and destroy objects
- To create: start the lifeline at the message.
- To destroy: put a cross



Sequence diagram

Simple example



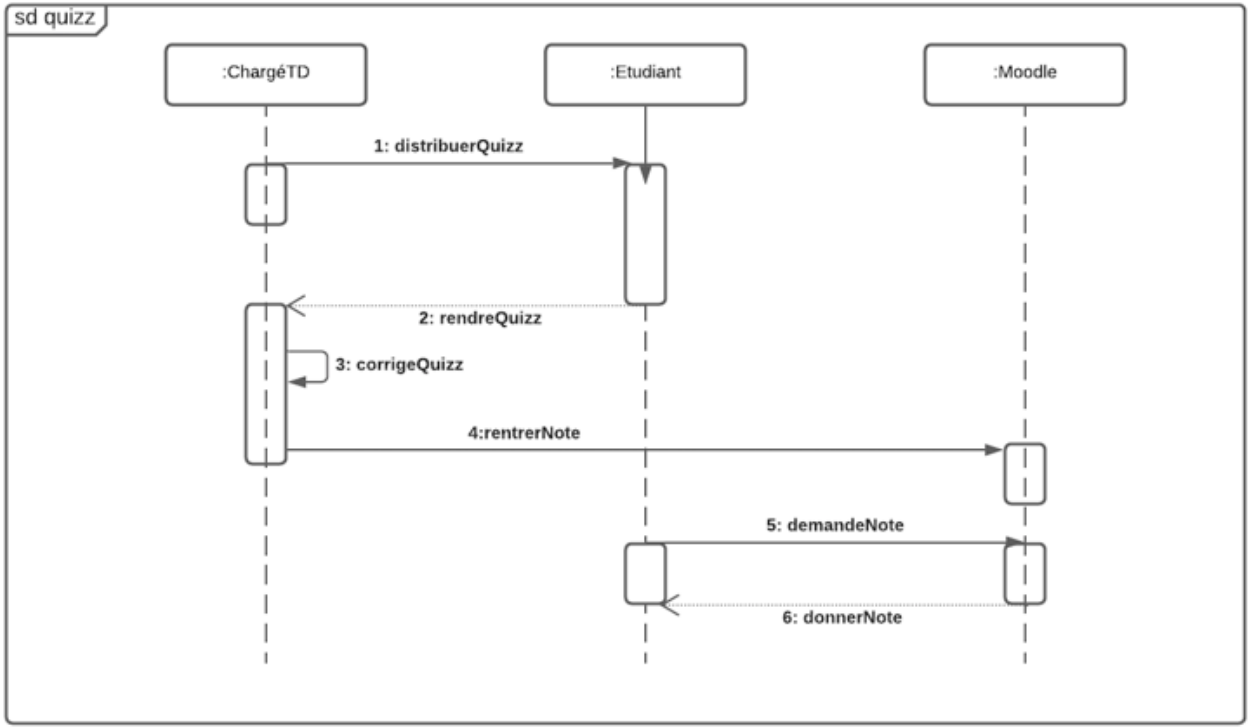
Sequence diagram

Organized sequence diagrams - reference

- As we have seen with the previous example, sequence diagrams can become cluttered.
- You can use ”Reference fragments” to logically organize your sequence diagrams.
- To define the sequence diagram to be used as a reference: box with “sd: name of the sequence diagram”
- sd = sequence diagram

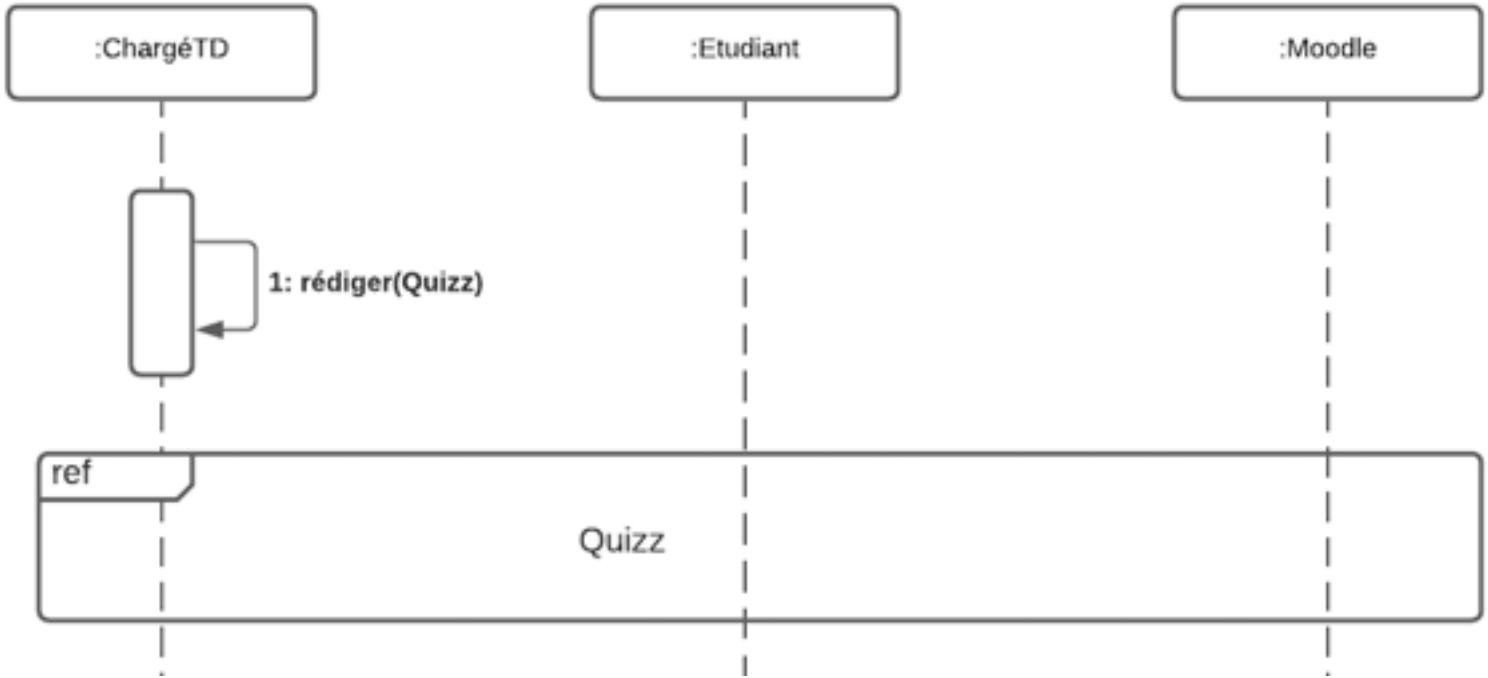
Sequence diagram

Organized sequence diagrams



Sequence diagram

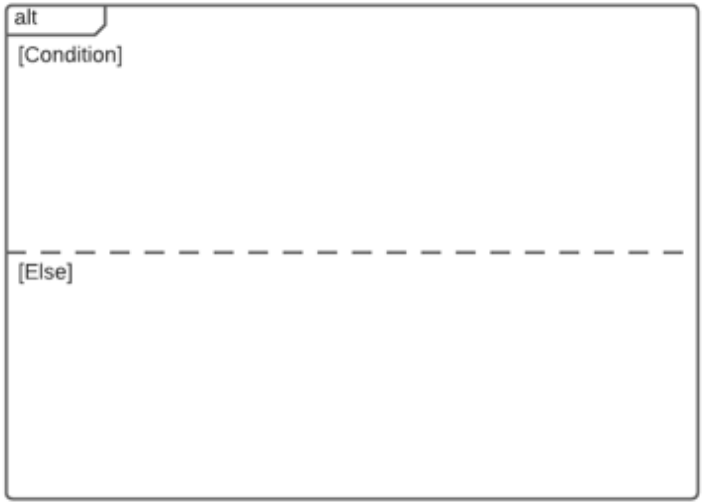
Organized sequence diagrams



Sequence diagram

Organized sequence diagrams - alternatives

- Possible to define alternatives, in the form of ”if then else”
- Note: possible to have else ifs (through added dashed lines)



Sequence diagram

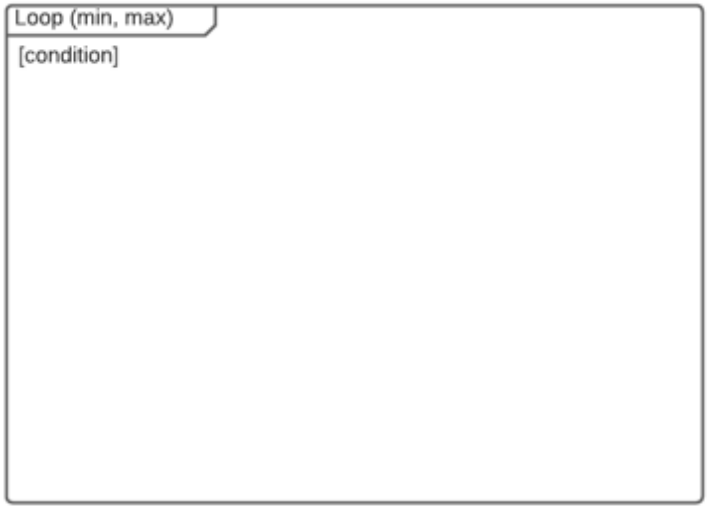
Organized sequence diagrams - option

- Possible to define option, in the form of ”if then”

Sequence diagram

Organized sequence diagrams - loops

- Possible to use loops, between two numbers (min and max) and/or until condition is true



Sequence diagram

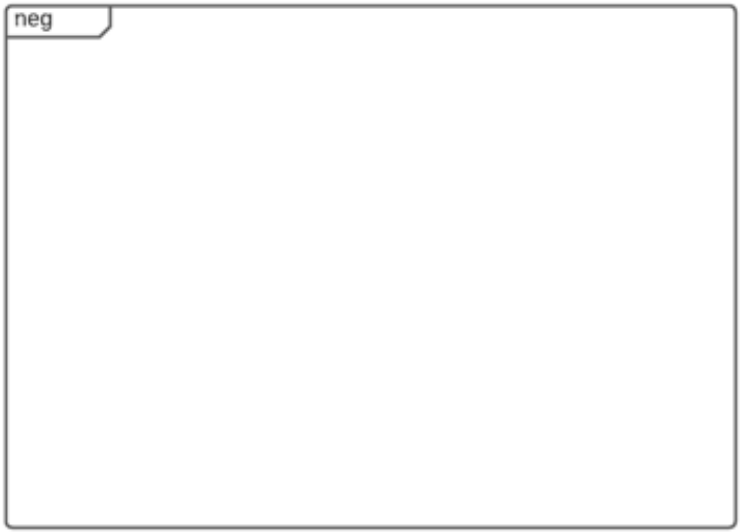
Organized sequence diagrams - break

- When condition is true:
  - Runs instruction inside break module
  - leave the fragment containing the break module

Sequence diagram

Organized sequence diagrams - Negation

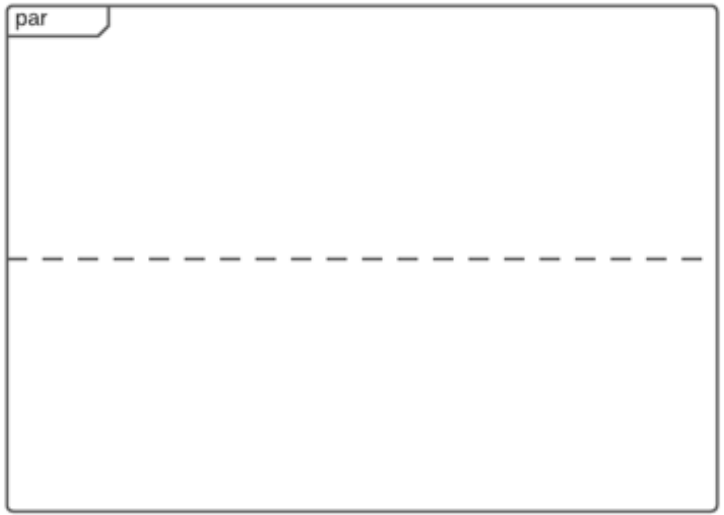
- Defines a sequence that is strictly forbidden



Sequence diagram

Organized sequence diagrams - Parallelism

- Defines two fragments that are runs simultaneously



Sequence diagram

Organized sequence diagrams - Critical

- Defines a fragment that cannot be interrupted

UML to model dynamic

Conclusion

- Sequence diagrams are useful to:
  - show how objects are interacting with each others
  - it can also be useful, by refining the objects, to “discover” new objects and their methods
- They can become quickly complex.
- Should refer to a use case.