



**POLITECNICO**  
MILANO 1863

# Software Engineering 2

UML and Requirements Engineering

Airbus example with UML



# Requirements Engineering (RE)

UML notation in RE

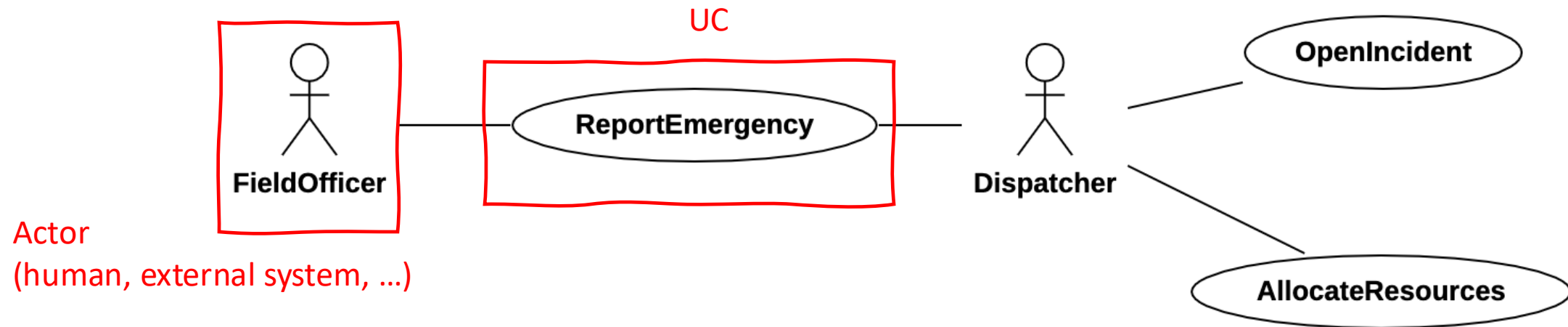
# Dynamic modeling



# Use Cases and UML

- **Use case**
  - flow of events in the system, including interaction with actors
  - It is initiated by an actor
  - Each use case has a name
  - Each use case has a termination condition
- **UML Use Case Model** (or diagram)
  - Set of **all use cases** specifying the complete functionality of the system

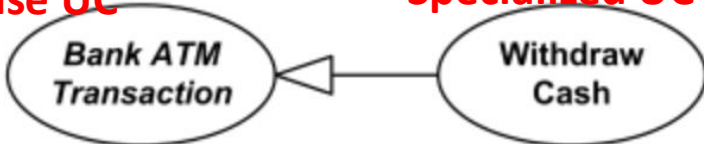

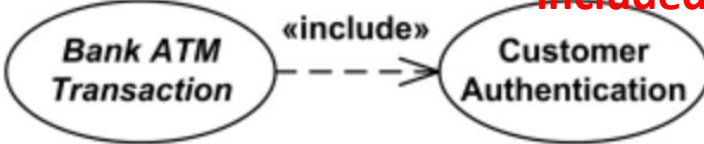
# Use Case Model: Incident Management System Example



# Use Case Associations

- Use case **association**: relationship between use cases
- Important **types** of associations
  - **Include**
    - A use case uses another use case (functional decomposition)
  - **Extend**
    - A use case extends another use case
  - **Generalization**
    - An abstract use case has several different specializations

# Use case relationships compared

Generalization	Extend	Include
<p><b>Base UC</b>      <b>Specialized UC</b></p> 	<p><b>Extending UC</b></p> 	<p><b>Included UC</b></p> 
Base use case could be <b>abstract use case</b> (incomplete) or concrete (complete).	Base use case is complete (concrete) by itself, defined independently.	Base use case is incomplete ( <b>abstract use case</b> ).
Specialized use case is required, not optional, if base use case is abstract.	Extending use case is optional, supplementary.	Included use case required, not optional.
No explicit location to use specialization.	Has at least one explicit extension location.	No explicit inclusion location but is included at some location.
No explicit condition to use specialization.	Could have optional extension condition.	No explicit inclusion condition.

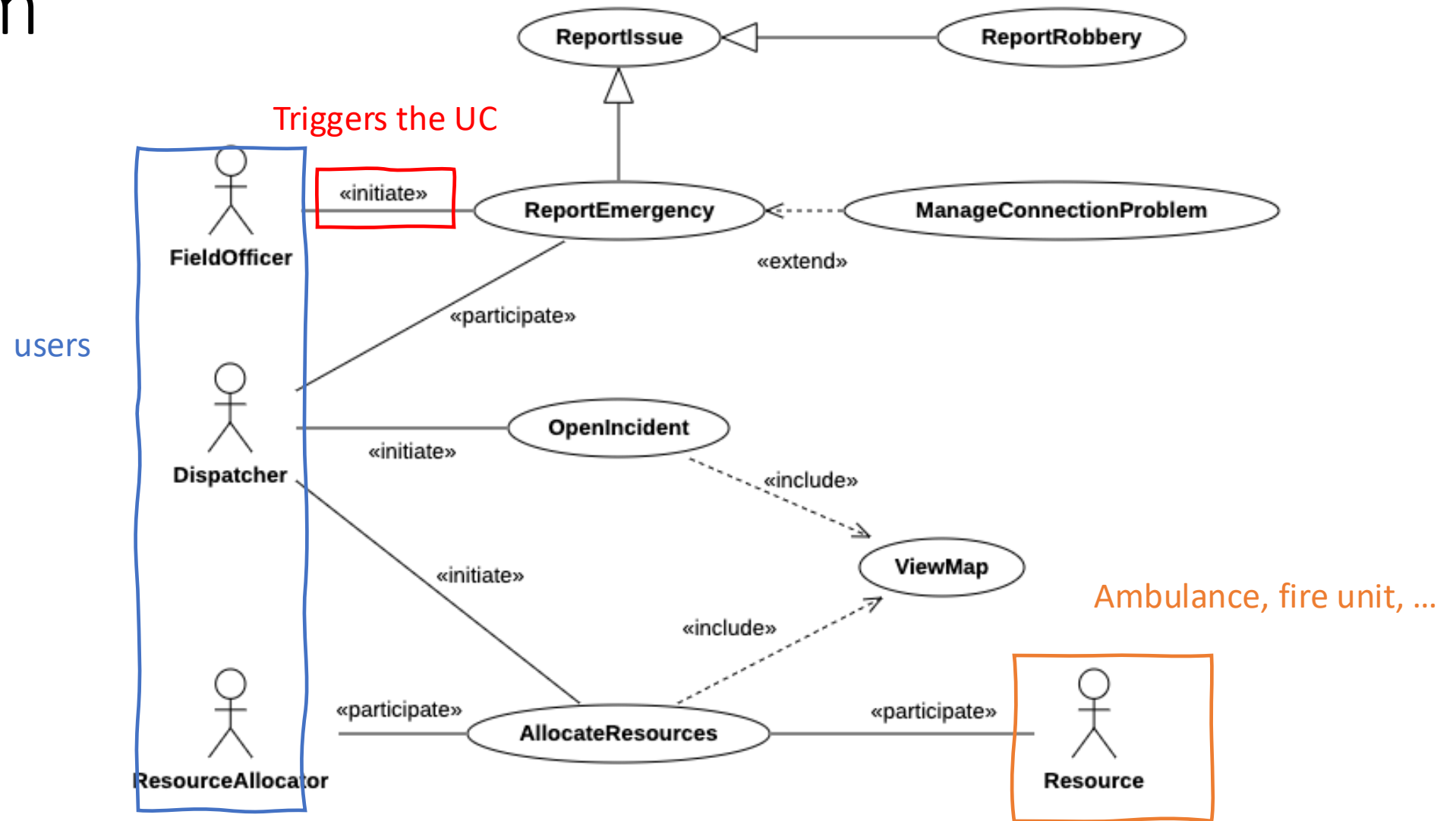
Extend: <https://www.uml-diagrams.org/use-case-extend.html>

Include: <https://www.uml-diagrams.org/use-case-include.html>

# Example: refined Incident Management System



POLITECNICO  
MILANO 1863





# Requirements-level class diagrams

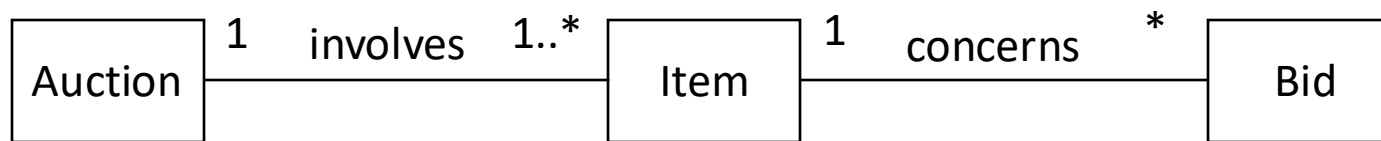
- They are **conceptual models** for the application **domain**
  - different from OO software design models
- They may include entities that will not be represented in the software-to-be
  - Implementation follows requirement modelling =>
  - Implementation details still unknown
- Usually, **no operations** (methods) of entities are modelled
  - Sometimes it is useful to introduce high-level functions of the system
  - Postpone decisions on detailed operations to software design

# Exercise (from WE2 exam of Feb 14, 2022)

- We have to develop a system, AuctionManager, for the management of auctions of items (e.g., paintings, pieces of art, memorabilia, etc.).
- Each auction involves a batch of items, which are bid upon one by one. For each successive item, the auction director opens the bidding; then, participants in the auction can bid on the item; if no new bid arrives within 4 minutes since the last one, the bidding closes, and the highest bid wins the right to buy the item (notice that each new bid must be higher than the previous one).
- AuctionManager must handle two types of participants to the auction: people who bid remotely (who do not physically attend the auction), and people who bid in person, by physically attending the auction. In the case of people who bid in person, the bidder raises a panel with an id of the bidder and the amount bid; in this case the bidding is tracked by an auction assistant (possibly the auction director her/himself), who records the bid in the application. Whenever a new bid arrives (either in person, or remotely), within 1 second the new highest bid is shown both to remote bidders and on a panel, physically, in the auction room.
- The system allows auction organizers to set up the auction, by inserting/modifying the information about the auctioned items and the order in which the items are going to be shown in the auction.
- The system also allows third-party applications to access the information about future and past auctions, to allow interested stakeholders to build statistics regarding auctions (e.g., determine which items are most likely to draw interests from bidders), and to advertise auctions.

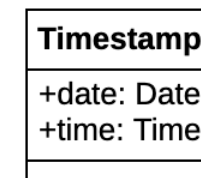
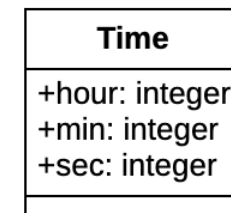
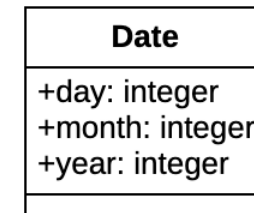
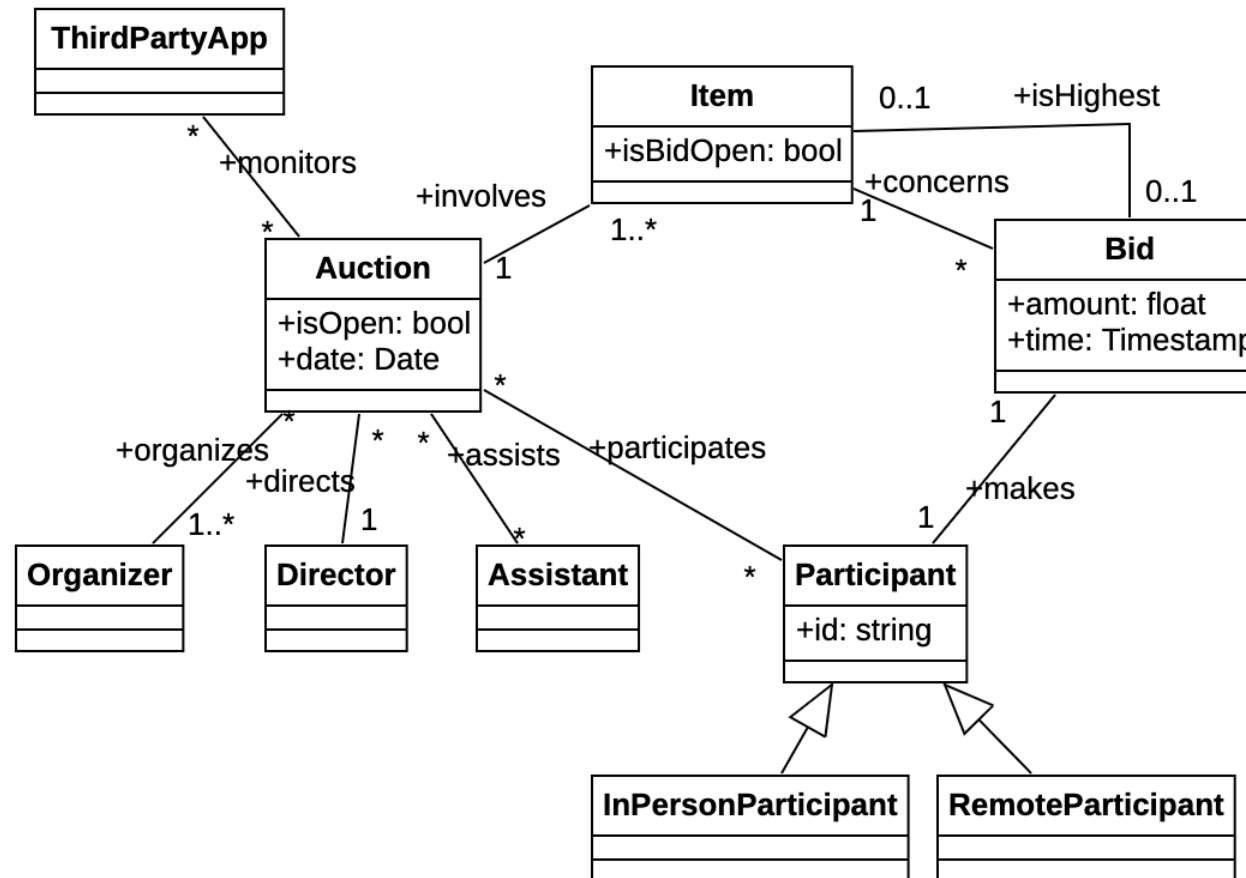
# Exercise (from WE2 exam of Feb 14, 2022)

- How do I derive a model of the domain?
- **Noun-verb analysis**
  - **Nouns** may represent: domain entities (classes), specializations (subclasses), fields, data (attributes)
  - **Verbs** may represent: associations between classes, operations
- “Each **auction** **involves** a batch of **items**, which **are bid** upon one by one...”



- Note that bid is not defined here as an association based on its other characteristics defined in the description (the bid corresponds to an amount of money, is placed by someone...)

# Exercise possible solution: Domain-level Class Diagram



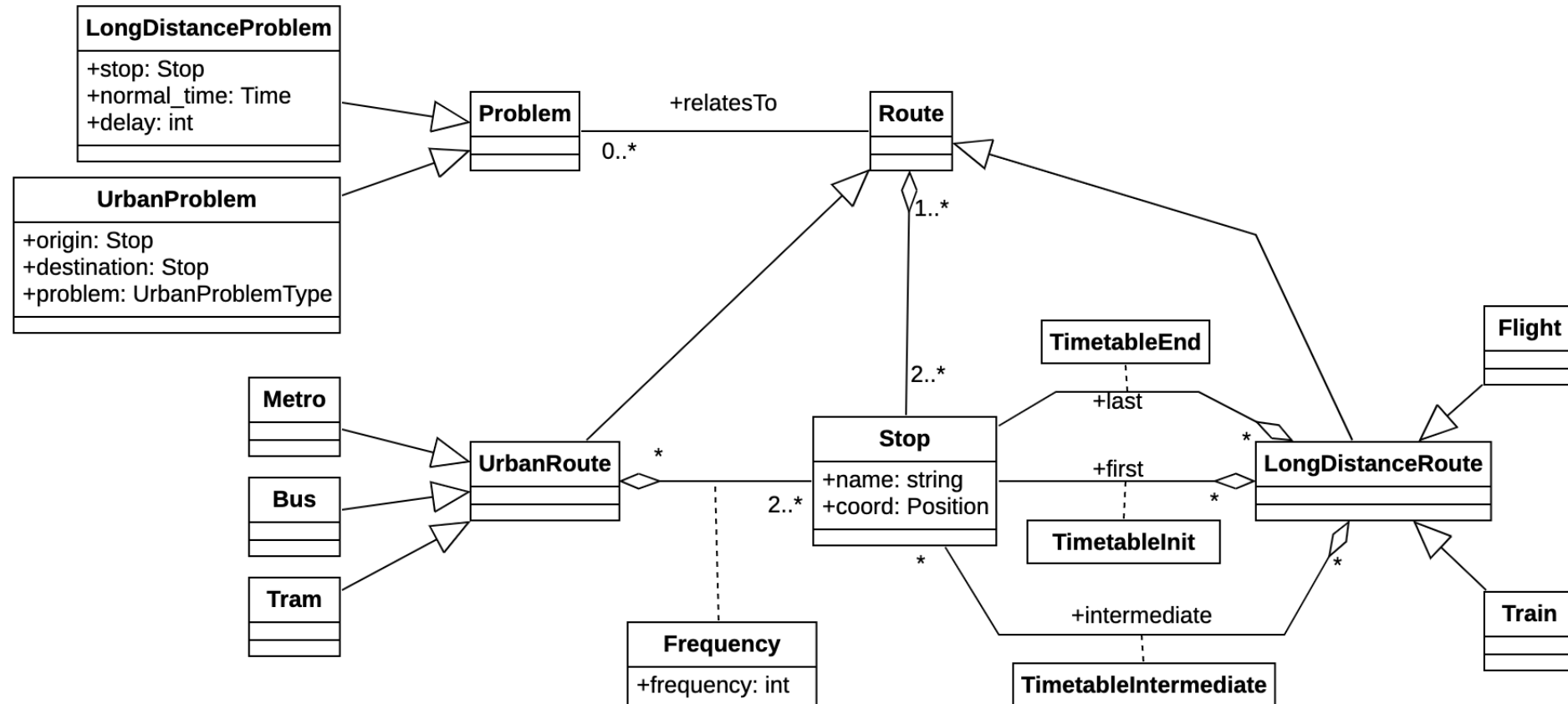
# Example: Trip Tracker

- Consider a system that warns travellers about problems on routes of their interest. The system monitors various kinds of transportation means and it sends users notifications whenever it detects problems along the monitored routes. Notifications are sent only to those users that showed an interest in the problems related to those routes.
- The system monitors both long-distance routes (intercity trains, flights) and urban public transports (metro, bus, trams). Each route has an identifier (for example, the number of the metro line, or the flight or train number), and a set of stops. Each stop has a name and geographical coordinates. In case of long-distance routes, each stop has an arrival time and a departure time; in case of urban routes, each line is associated with a frequency of stops (let us assume, for simplicity, that the frequency is the same throughout the day, e.g., “every 5 minutes”).

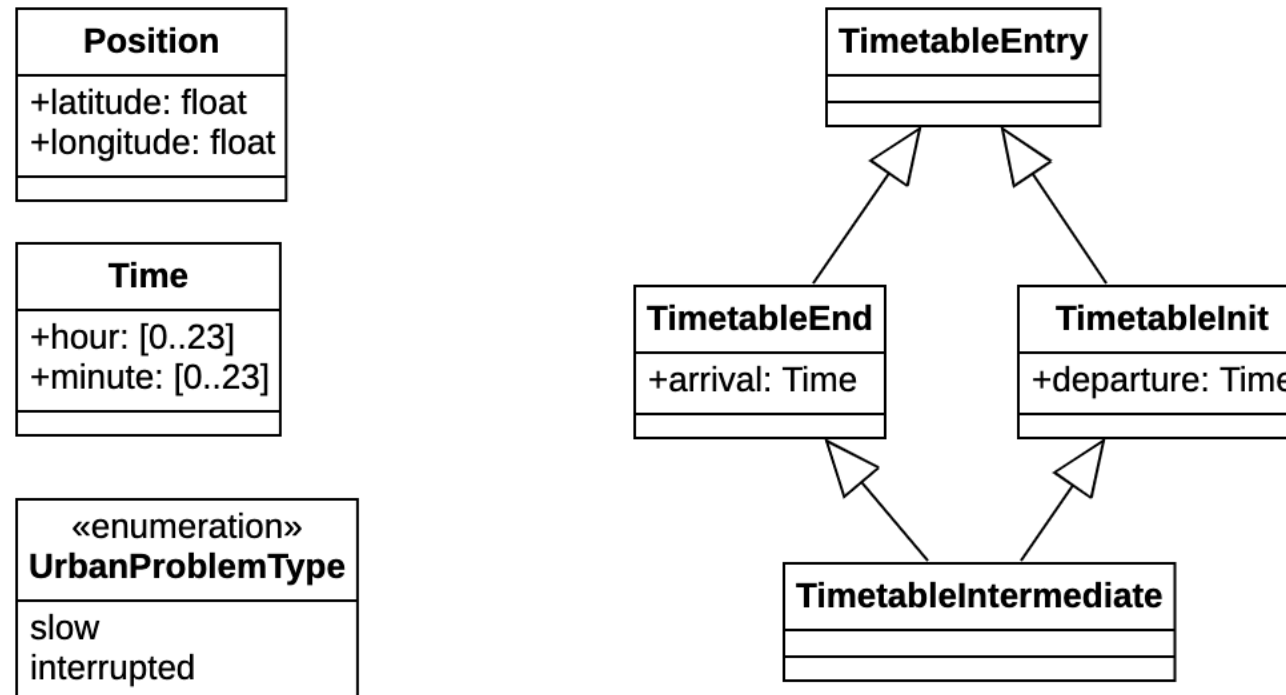
# Trip Tracker (2)

- The system manages different kinds of problems, depending on the monitored means of transportation. In case of long-distance routes, the detected problems are the expected delays at the various stops. More precisely, the information associated with the notification of a problem in this case include the id of the route, the stop, the nominal arrival time, and the expected delay. In the case of urban transport, instead, the monitored problems are the interruptions or the slowdowns along the routes. In this case, each notification includes the id of the affected line, the segment (identified by the departure stop and the arrival stop) in which the problem arose, and the problem description (which for simplicity can only be a slowdown or an interruption).
- The system allows users to “subscribe” to notifications of problems of interest, in order to receive them. Each user has an identifier, and to subscribe they only need to tell the system their own id and the id of the route for which the user is interested in receiving notifications. Each user can also eliminate a previous subscription, by indicating their id and the id of the route in which the user is no longer interested. When the system detects a problem concerning a route to which a user has subscribed, it sends the latter a notification with the information associated with the problem.

# Domain-level Class Diagram (1)

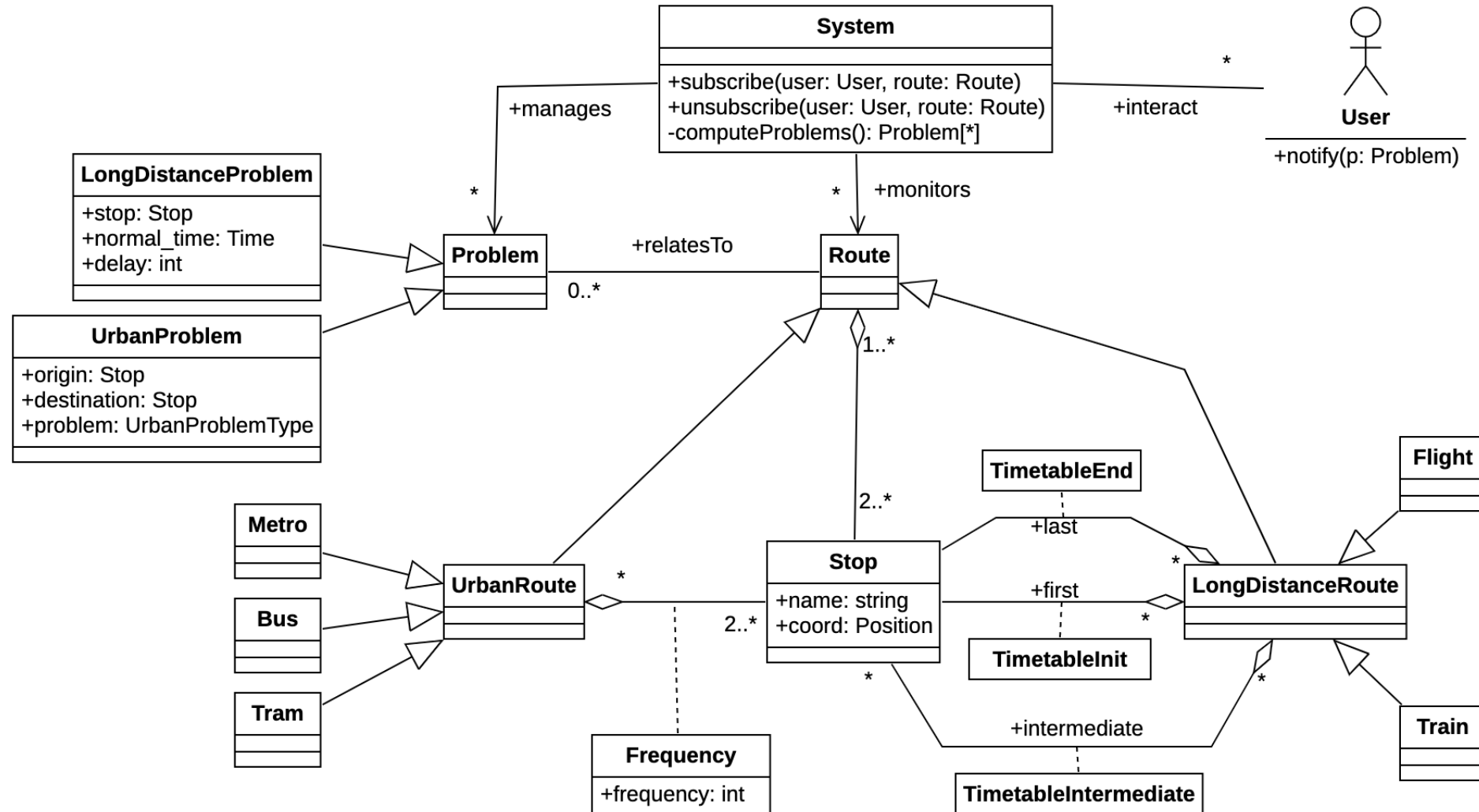


# Domain-level Class Diagram (2)





# Domain-level Class Diagram (with system)



# Dynamic modeling

- **Purpose**
  - Model interactions, behaviors of participants and workflow
- How do we do this?
  - Consider use cases or scenarios
  - Model interaction between objects → **sequence** diagram
  - Model dynamic behavior of a single object → **state machine** diagram (“state diagram” for short)
  - Model workflow → **activity** diagram

# Sequence diagrams

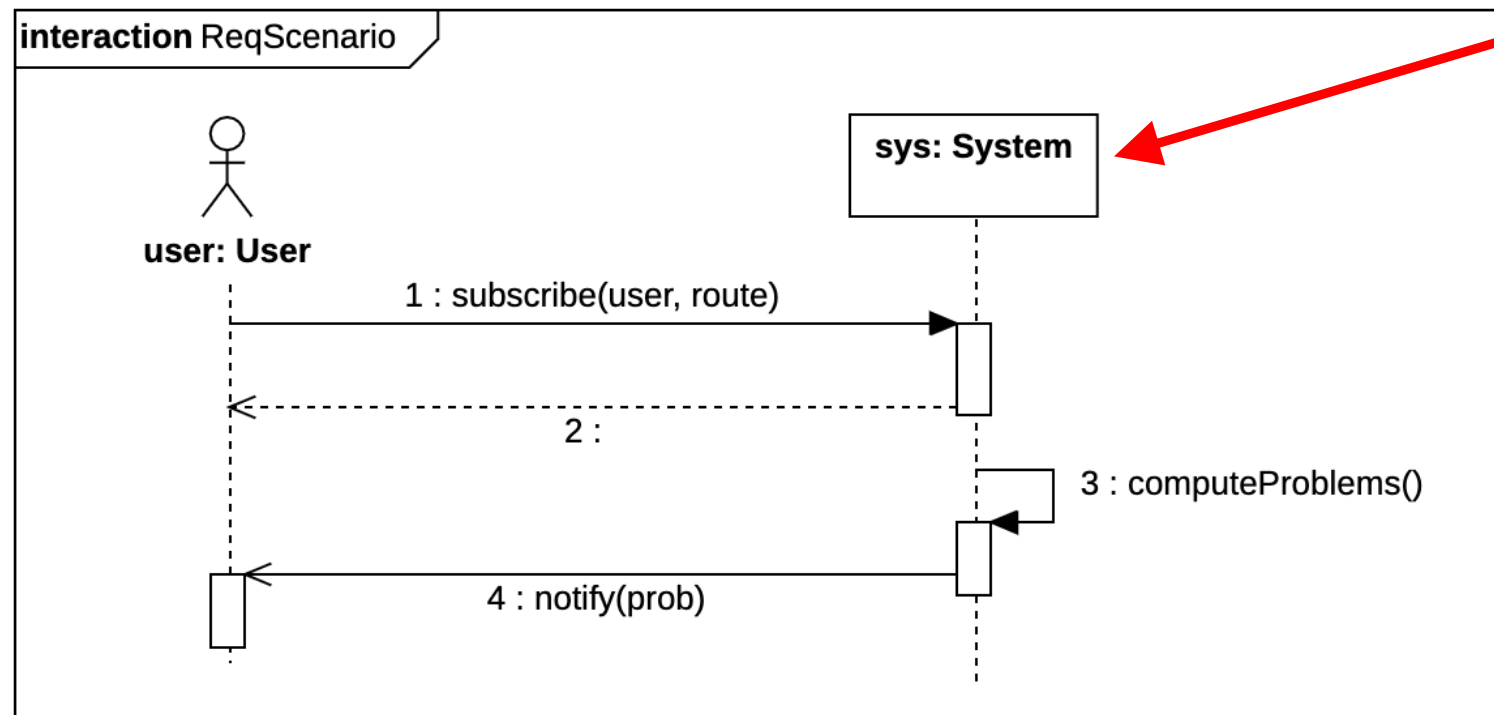
- **Sequence diagram**: graphical description of objects participating in a use case or scenario using a DAG (directed acyclic graph) notation
- **Steps**
  - Analyze the flow of **events** in a use case (or scenario)
  - An event has a **sender** and a **receiver**
  - The representation of the event is referred to as **message** in the sequence diagram notation
  - Sender and receiver are **objects** participating in the use case
- **Note**
  - Objects/classes have already been identified during object modeling
  - Other objects may emerge from dynamic modelling
  - Different views must be consistent

# Example of requirement-level SD (from Trip Tracker)

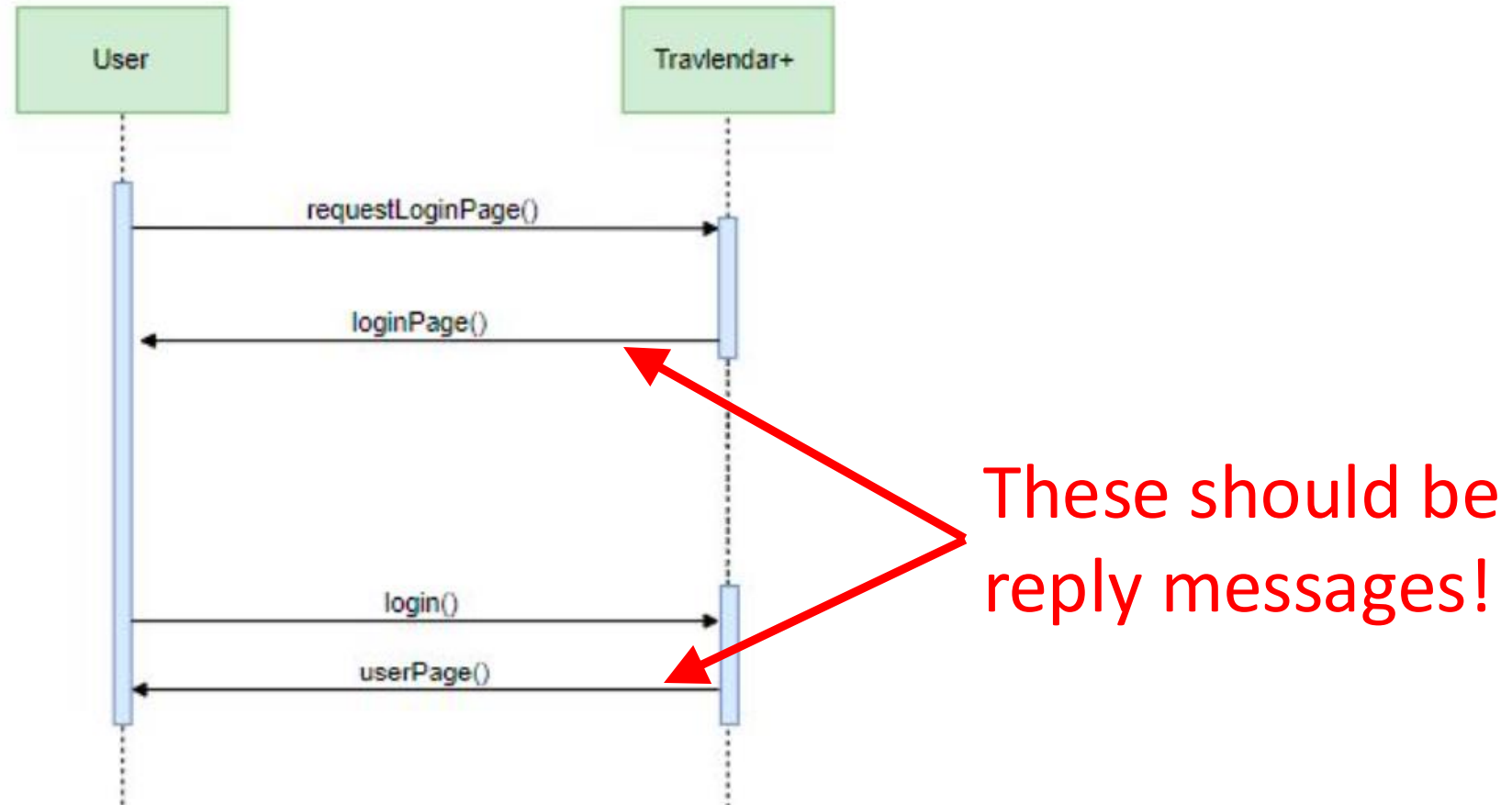


POLITECNICO  
MILANO 1863

This is the S2B!

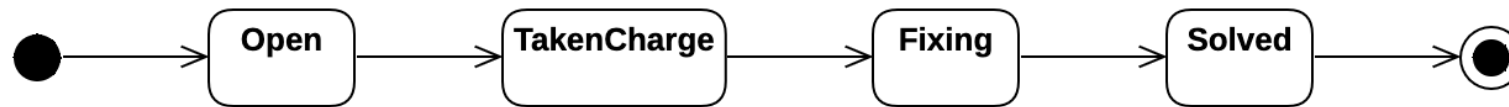


# An example of incorrect sequence diagram

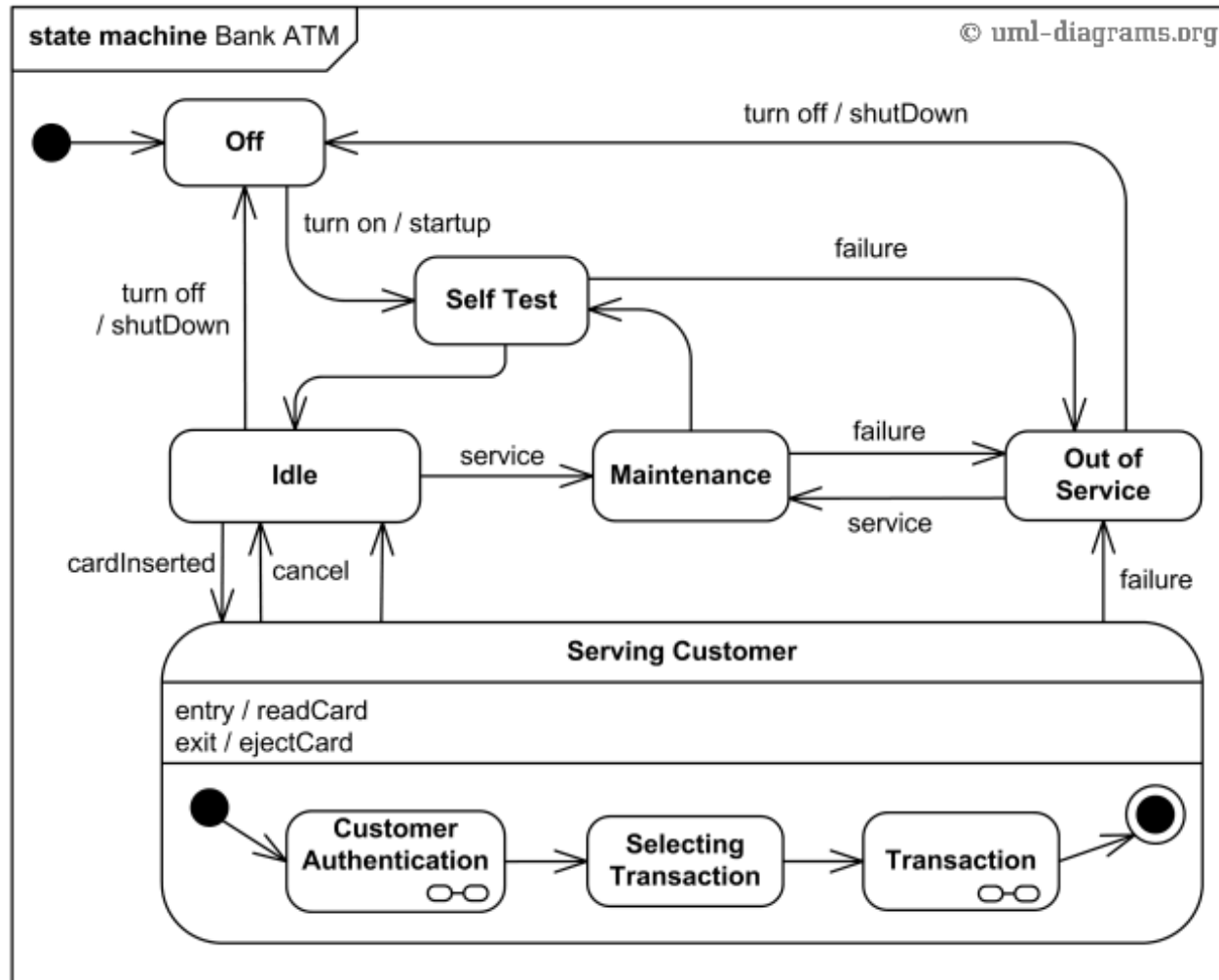


# State Machine diagrams

- **State machine** diagram: captures the behavior of the objects that are instances of a certain class
- Example: State Machine associated with Problem in Trip Tracker Example



# Another example of state diagram



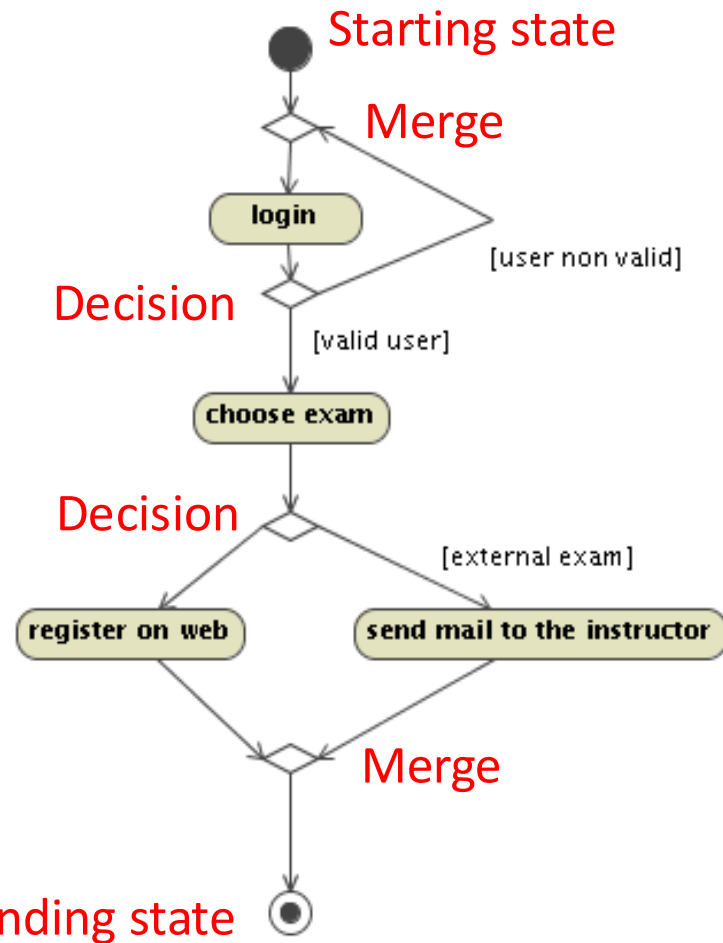
# State diagram vs sequence diagram

State diagram	Sequence diagram
Focus on changes in an individual object over time	Focus on the interaction between objects over time
Class-level documentation	Instance-level documentation
We can infer many possible event and order-dependent behaviors	Shows one specific case or a subset of cases

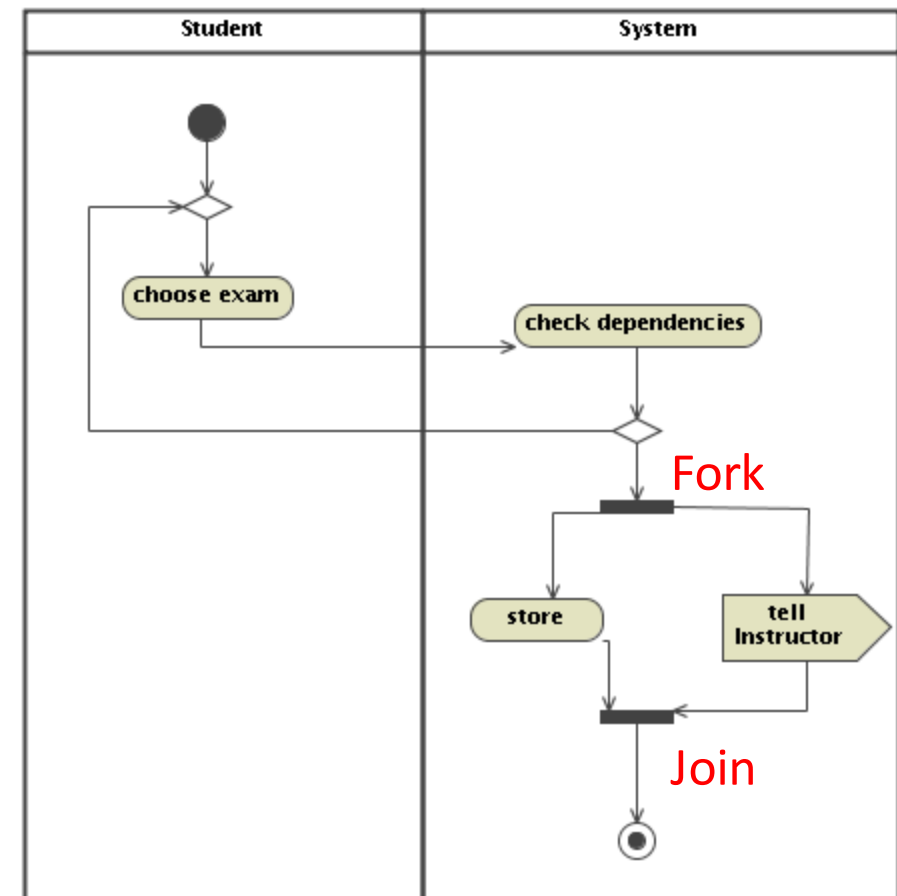


# Activity diagrams

Describes an activity without assigning responsibilities



Describes “register on the web” by highlighting the responsibilities of the student and of the system



# Sequence diagram vs activity diagram

---

## Sequence diagram

Focus on object interaction

Suited for describing an interaction protocol

## Activity diagram

Focus on activities and flow of activities

Suited for describing a process

---

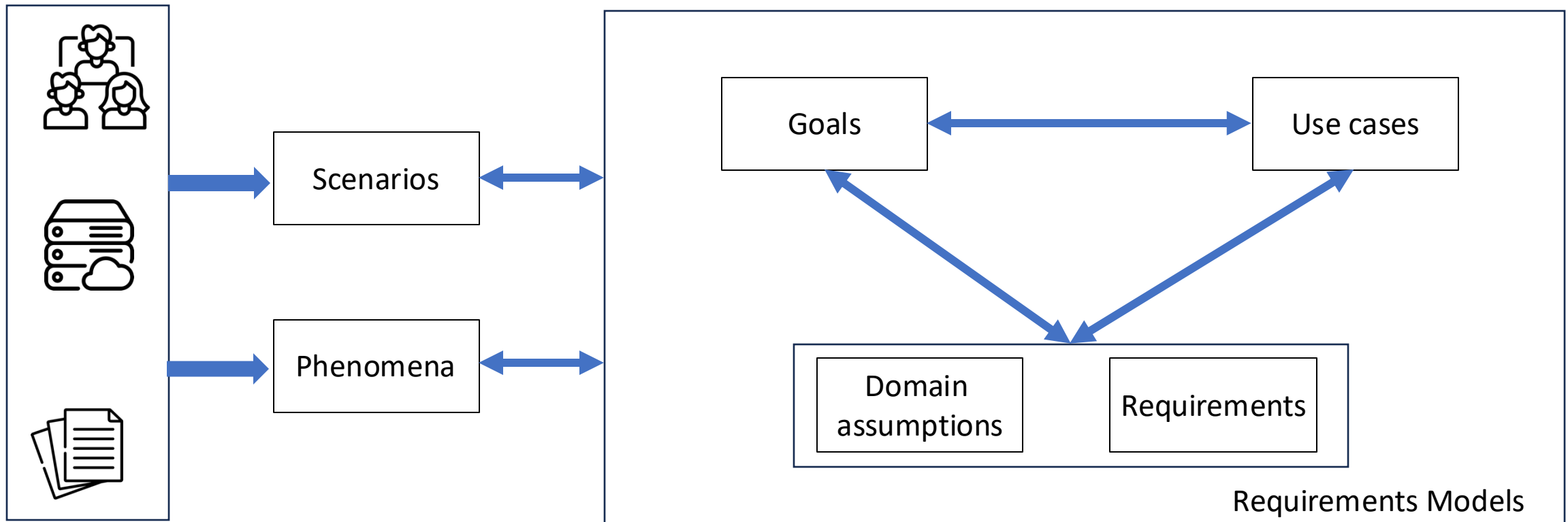
# Summary: UML for RE

- What are the input/output transformations?
  - Create **use case diagrams** (and use cases) from scenarios (functional modeling)
- What is the structure of the world?
  - Create **class diagrams** (static view)
    - Identify domain entities
    - What are the associations between them?
    - What is their multiplicity?
    - What are the attributes of the objects?
    - What high-level operations are defined on the objects?
  - Is there any state change in an object that is to be defined explicitly? If yes, create **state diagrams** (dynamic view — class behavior models)

# Summary: UML for RE

- How is the expected interaction S2B-environment?
  - Create **sequence diagrams** from use cases (dynamic view — object behavior examples)
    - Identify event senders and receivers
    - Show sequence of events exchanged between objects
    - Identify event dependencies and event concurrency
  - Create activity diagrams when you want to highlight important processes (dynamic view — workflow)

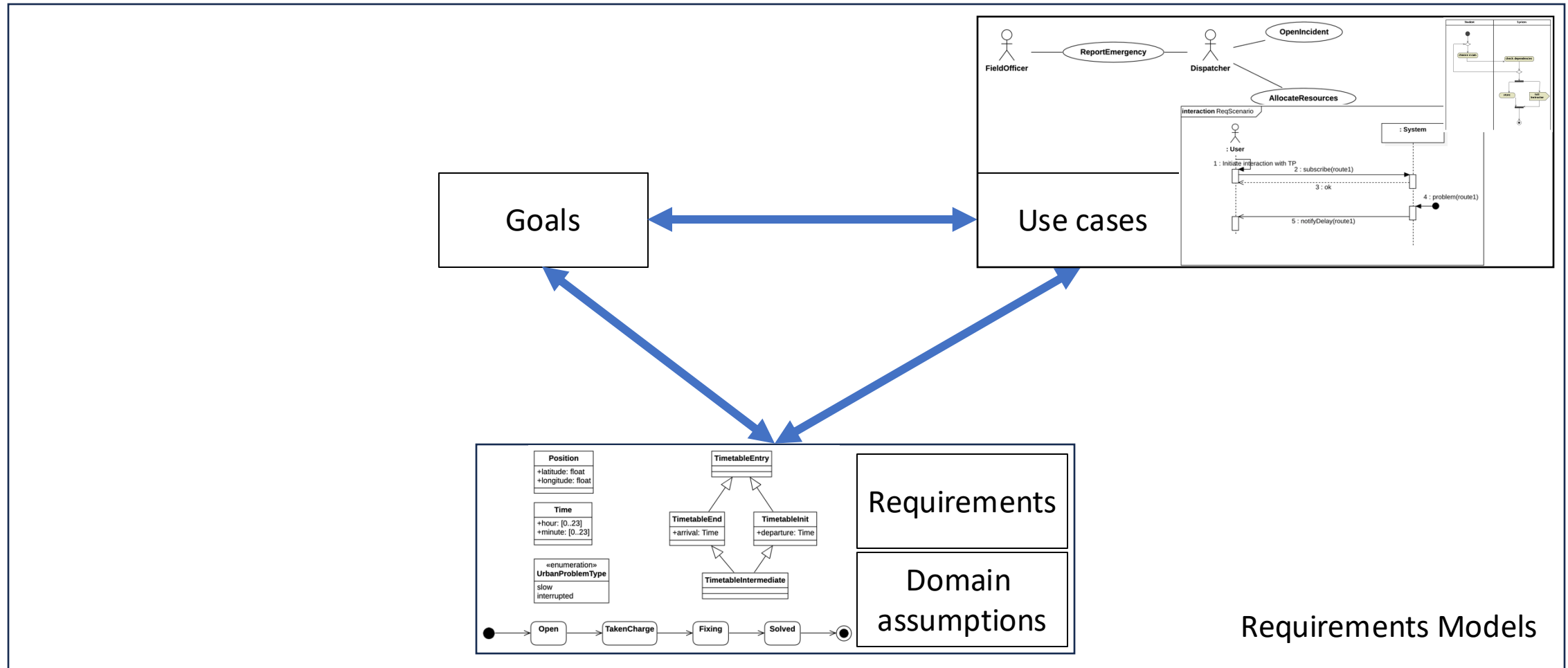
# Req elicitation and modeling



# Req elicitation and modeling: Adding the new ingredients



POLITECNICO  
MILANO 1863

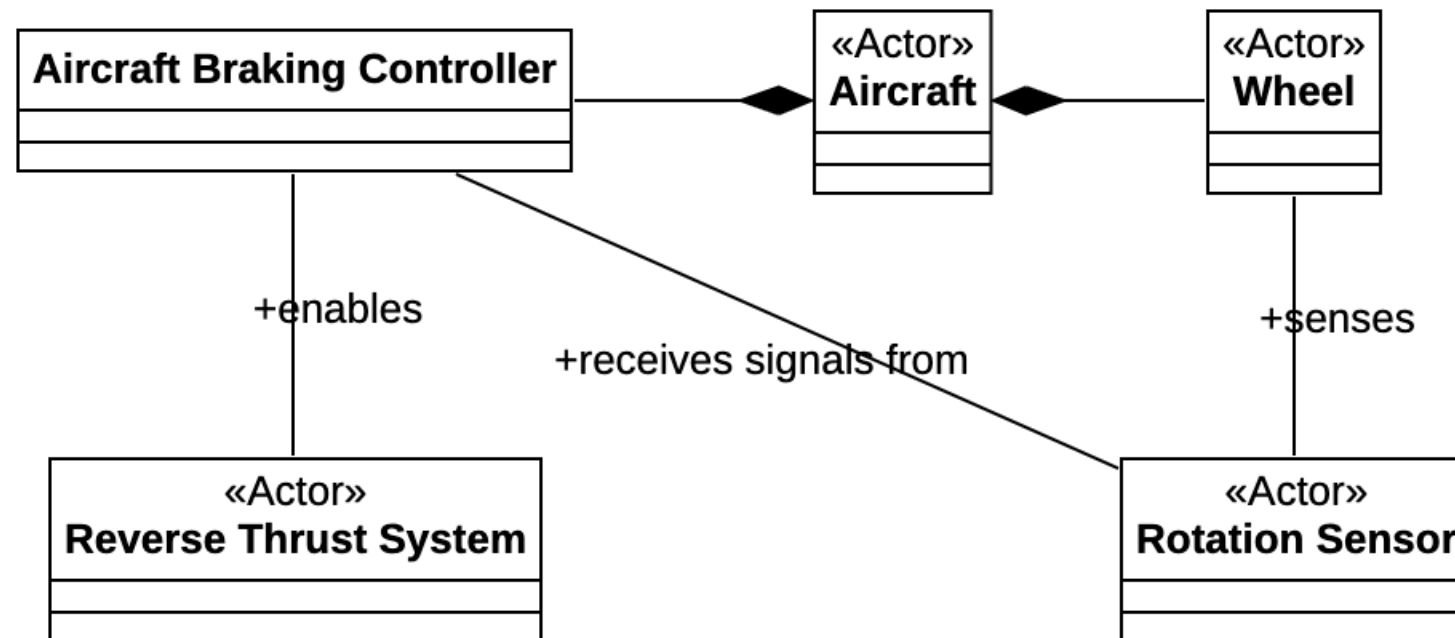




# Requirements Engineering (RE)

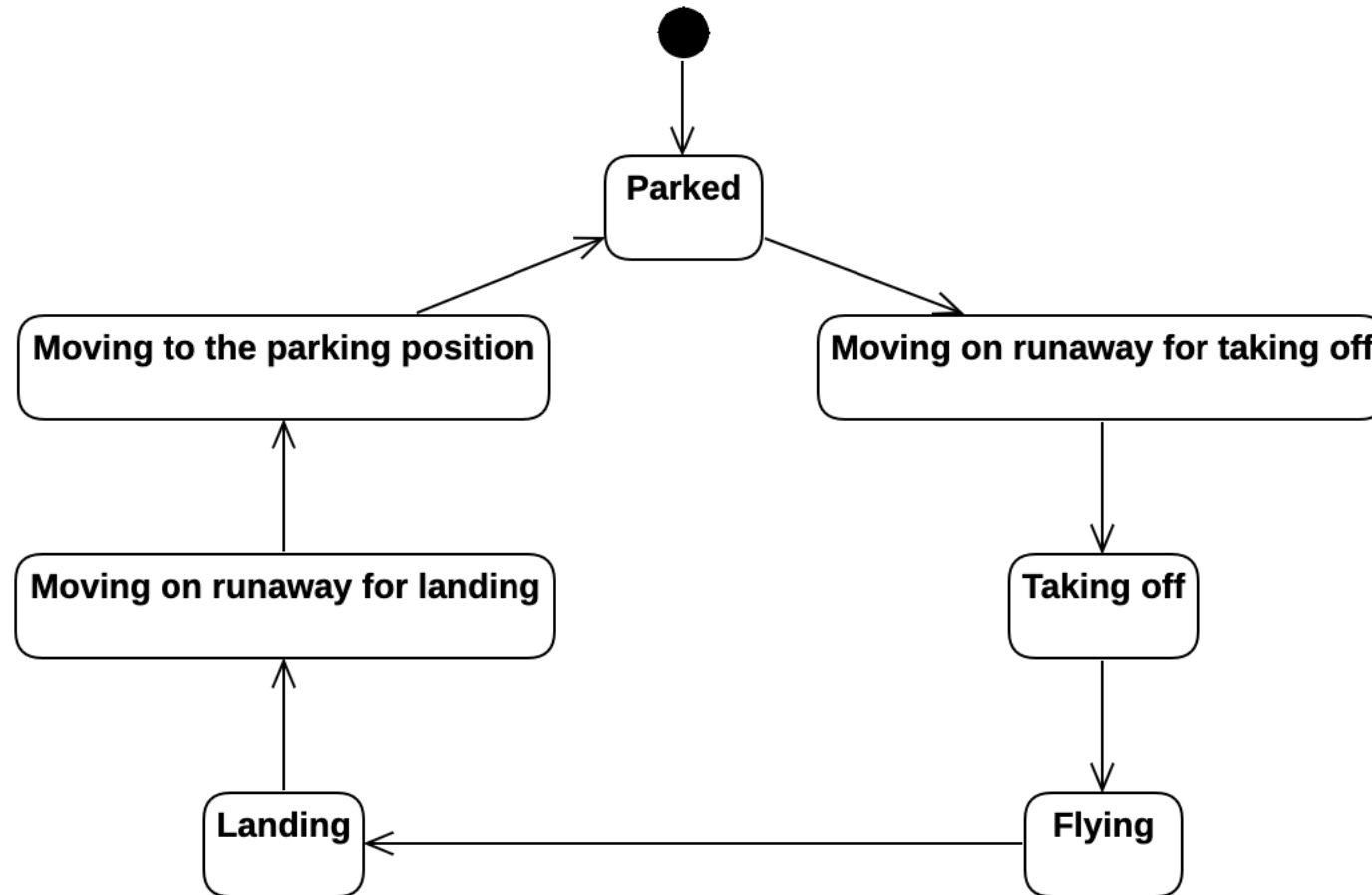
Modeling the Airbus braking logic with UML — What can we model? What not?

# Domain modeling: main entities and relationships

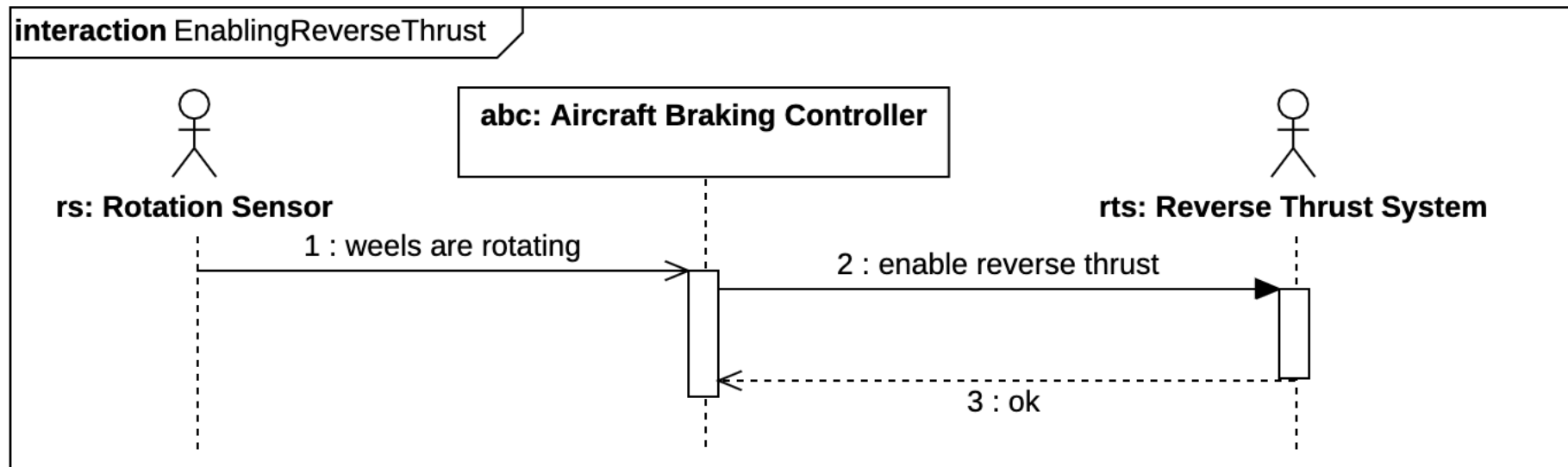
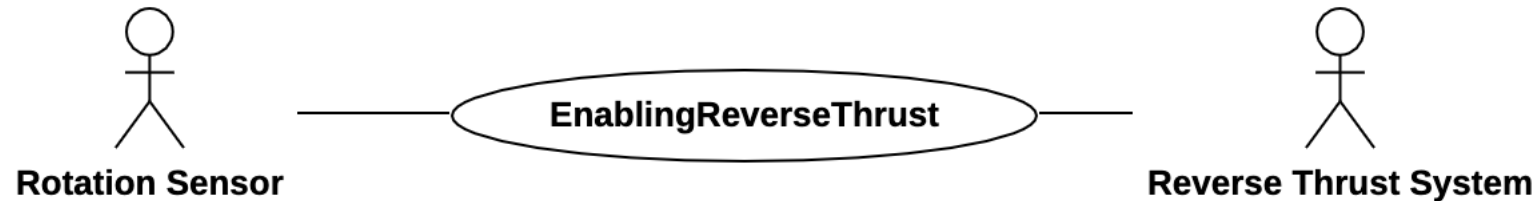




# Domain modeling: states of an aircraft



# Dynamic behavior: enabling reverse thrust when wheels are rotating



# Is the UML spec complete?

- We have described
  - All phenomena: Aircraft, wheels, sensor, reverse thrust system
  - A use case EnablingReverseThrust
- Are we missing something?
- Are we representing goals, domain properties and requirements?
  - Goal
    - $\text{Reverse\_enabled} \Leftrightarrow \text{Moving\_on\_runway}$
  - Domain properties
    - $\text{Wheel\_pulses\_on} \Leftrightarrow \text{Wheels\_turning}$
    - $\text{Wheels\_turning} \Leftrightarrow \text{Moving\_on\_runway}$
  - Requirements
    - $\text{Reverse\_enabled} \Leftrightarrow \text{Wheels\_pulses\_on}$



# Is the UML spec complete?

- Pure UML does not help us in expressing assertions
- UML models must be complemented with some formal or informal description of these assertions

# References and interesting sources

- M. Jackson, P. Zave, "Deriving Specifications from Requirements: An Example", Proceedings of ICSE 95, 1995
- M. Jackson, P. Zave, "Four Dark Corners of Requirements Engineering", TOSEM, 1997
- B. Nuseibeh, S. Easterbrook, "Requirements Engineering: A Roadmap", Proceedings ICSE 2000
- A. van Lamsweerde, Requirements Engineering: From System Goals to UML Models to Software Specifications, Wiley and Sons, 2009
- M. Jackson, Software Requirements and Specifications: A Lexicon of Practice, Principles and Prejudices, ACM Press Books, 1995
- S. Robertson and J. Robertson, Mastering the Requirements Process, Addison Wesley, 1999
- Requirements Engineering Specialist Group of the British Computer Society <http://www.resg.org.uk/>
- B. Bruegge & A.H. Dutoit, Object-Oriented Software Engineering: Using UML, Patterns, and Java, 2nd Edition, Prentice Hall, Upper Saddle River, NJ, September 25, 2003
- T. E. Bell and T. A. Thayer. 1976. Software requirements: Are they really a problem?. In Proceedings of the 2nd international conference on Software engineering (ICSE '76). IEEE Computer Society Press, Los Alamitos, CA, USA, 61-68.
- F. P. J. Brooks, "No Silver Bullet Essence and Accidents of Software Engineering," in Computer, vol. 20, no. 4, pp. 10-19, April 1987. doi: 10.1109/MC.1987.1663532
- Slides by Emmanuel Letier UCL