Based on:
- Seidl et al.'s "UML@Classroom", Chapters 1 & 2
- Miles and Hamilton's "Learning UML 2.0", Chapter 1

# Introduction to UML

Software Design (40007) – 2024/2025

Justus Bogner
Ivano Malavolta

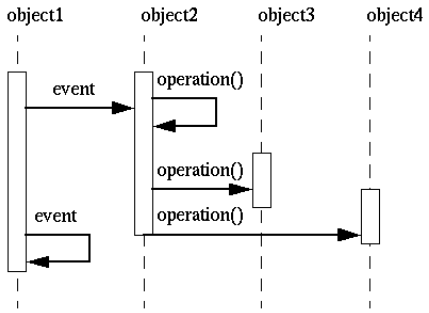
VU VRIJE UNIVERSITEIT AMSTERDAM

**Software and Sustainability research group (S2)**
Department of Computer Science, Faculty of Sciences

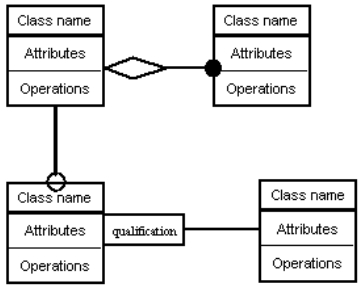
# What is UML?

- In the 80s there were multiple OO approaches
  - Each approach had its own notation
  - Rational Inc. (now **IBM**) tried to harmonize this (1994-1996)

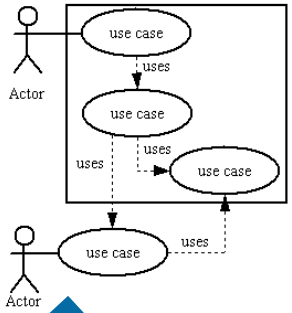
Booch notation




Jacobson's OOSE

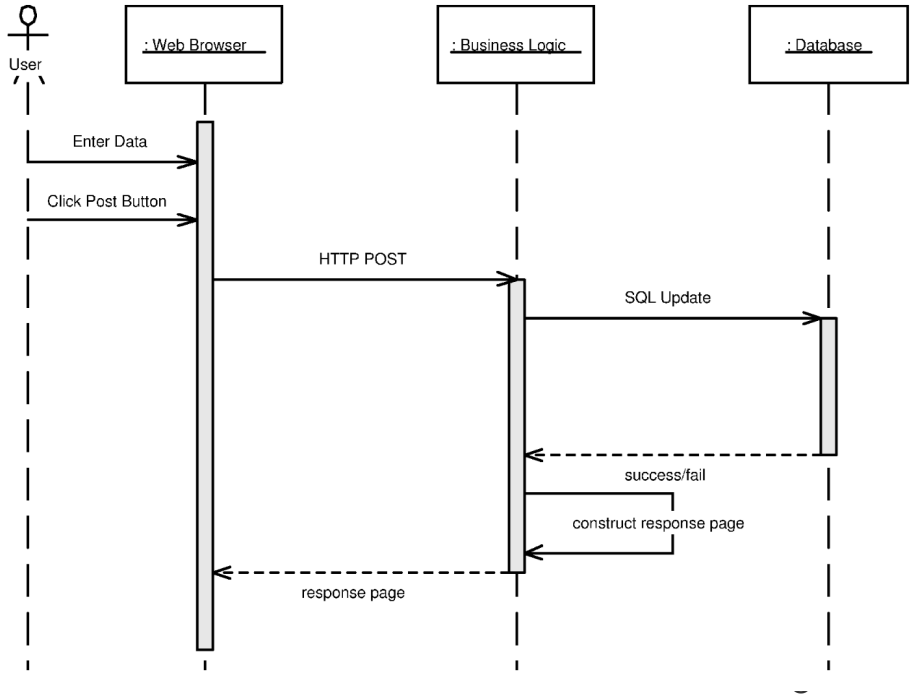



Rumbaugh's Technique



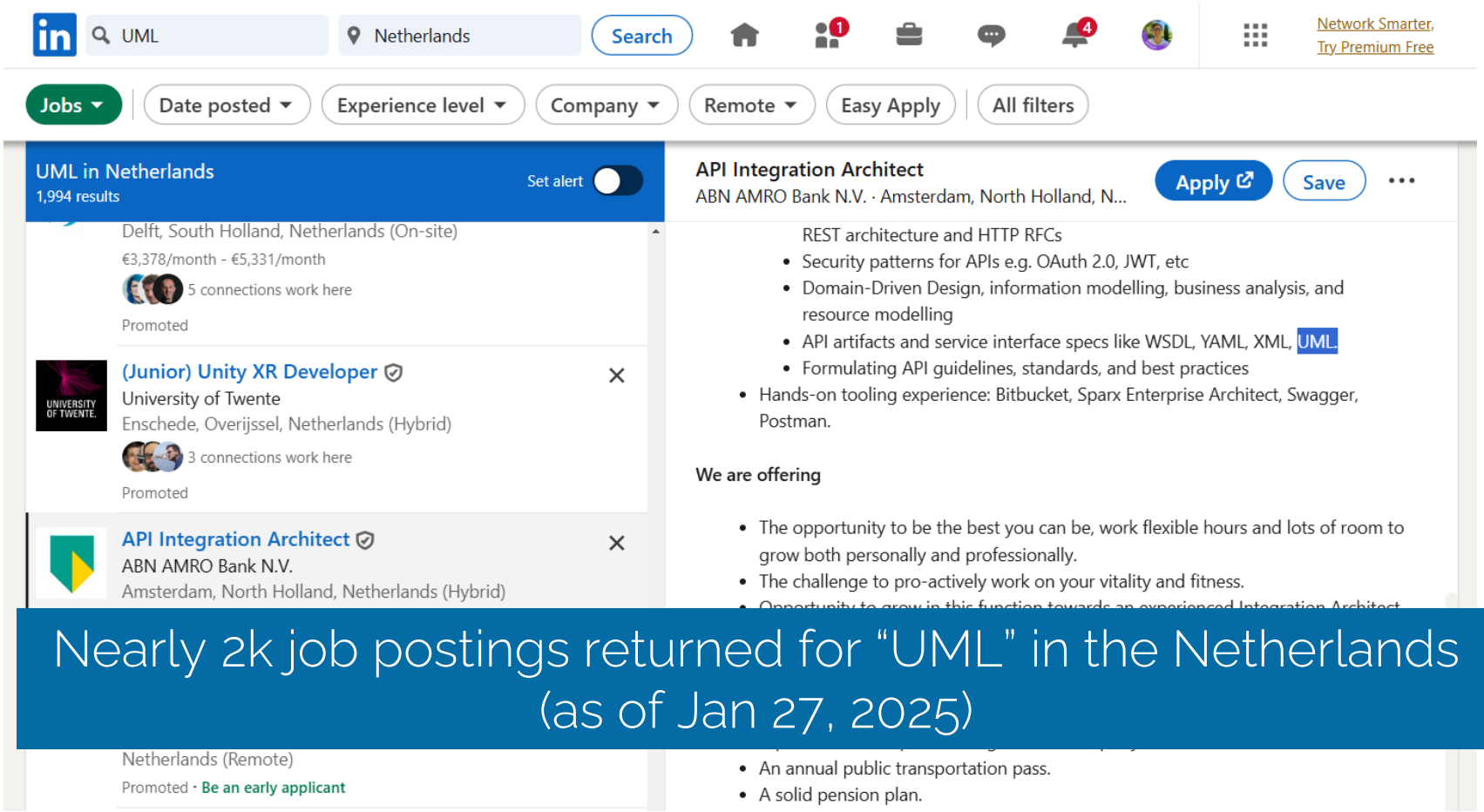

# What is UML?

- UML = Unified Modeling Language
- Standardized software design language
- Under the umbrella of Object Management Group (OMG)
- The “Swiss Army Knife” of notations

# Why UML in this course?

The most used language for modeling software




Nearly 2k job postings returned for “UML” in the Netherlands (as of Jan 27, 2025)



VU VRIJE UNIVERSITEIT AMSTERDAM

# Advantages of UML (1)

- It is not tied to **any development process**
  - Waterfall, Scrum, Kanban, XP, etc.
- Can be **used across the whole life cycle**
  - Promotes iterative refinement of models
- It is **scalable**
  - You can zoom in with additional details when needed
- It has **different representations**
  - Graphical
  - Textual
  - Machine-readable

# Advantages of UML (2)

- It is a **general-purpose modeling language**
  - It can be used for modeling a mobile app, but also a satellite
- It is **comprehensive**
  - Different parts of a system can be described with UML
- Supports both
  - **Descriptive** models (originally intended usage by its creators)
  - **Prescriptive** models (allows things like automatic model analysis and execution or code generation)

# Advantages of UML (3)

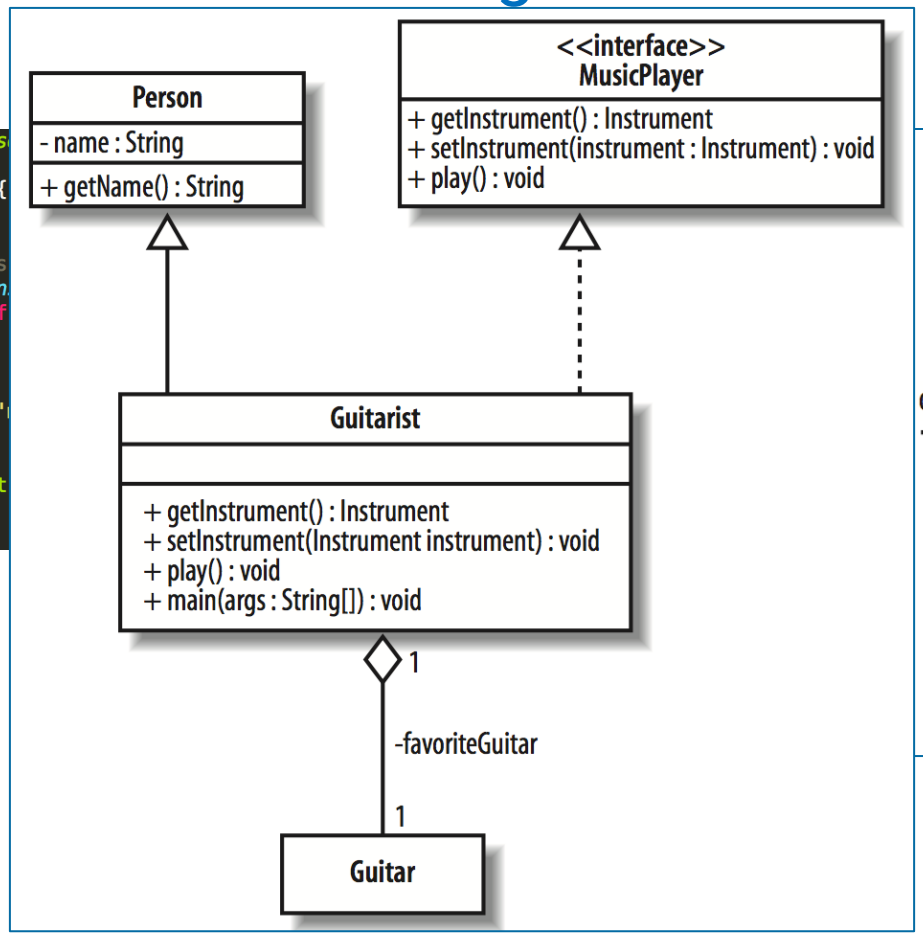
UML is a **semi-formal modeling language**

→Its core concepts have well-defined **semantics**

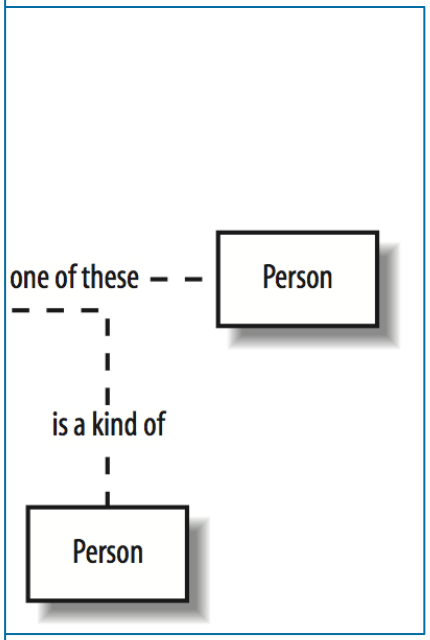
## Executable code

```
1▼ public class Guitarist extends Pers
2     Guitar favoriteGuitar;
3     public Guitarist (String name) {
4        super(name);
5▼    }
6        // A couple of local methods
7▼       public void setInstrument(In
8           if (instrument instanceof
9              this.favoriteGuitar =
10    }
11    else {
12             System.out.println("I'
13          }
14    }
15    public Instrument getInstrument
16    }
17    ...
```

## UML diagram




## Informal diagram






VU VRIJE UNIVERSITEIT AMSTERDAM

# Where are the “meanings” of UML concepts?

## The UML superstructure

640 pages like this →

Don't read it! Use it only as a manual in case of doubts

https://www.omg.org/spec/UML

### 16.3.6 UseCase (from UseCases)

A use case is the specification of a set of actions performed by a system, which yields an observable result that is, typically, of value for one or more actors or other stakeholders of the system.

**Description**

A UseCase is a kind of behaviored classifier that represents a declaration of an offered behavior. Each use case specifies some behavior, possibly including variants, that the subject can perform in collaboration with one or more actors. Use cases define the offered behavior of the subject without reference to its internal structure. These behaviors, involving interactions between the actor and the subject, may result in changes to the state of the subject and communications with its environment. A use case can include possible variations of its basic behavior, including exceptional behavior and error handling.

The subject of a use case could be a physical system or any other element that may have behavior, such as a component, subsystem or class. Each use case specifies a unit of useful functionality that the subject provides to its users, i.e., a specific way of interacting with the subject. This functionality, which is initiated by an actor, must always be completed for the use case to complete. It is deemed complete if, after its execution, the subject will be in a state in which no further inputs or actions are expected and the use case can be initiated again or in an error state.

Use cases can be used both for specification of the (external) requirements on a subject and for the specification of the functionality offered by a subject. Moreover, the use cases also state the requirements the specified subject poses on its environment by defining how they should interact with the subject so that it will be able to perform its services.

The behavior of a use case can be described by a specification that is some kind of Behavior (through its ownedBehavior relationship), such as interactions, activities, and state machines, or by pre-conditions and post-conditions as well as by natural language text where appropriate. It may also be described indirectly through a Collaboration that uses the use case and its actors as the classifiers that type its parts. Which of these techniques to use depends on the nature of the use case behavior as well as on the intended reader. These descriptions can be combined. An example of a use case with an associated state machine description is shown in Figure 405.

**Attributes**

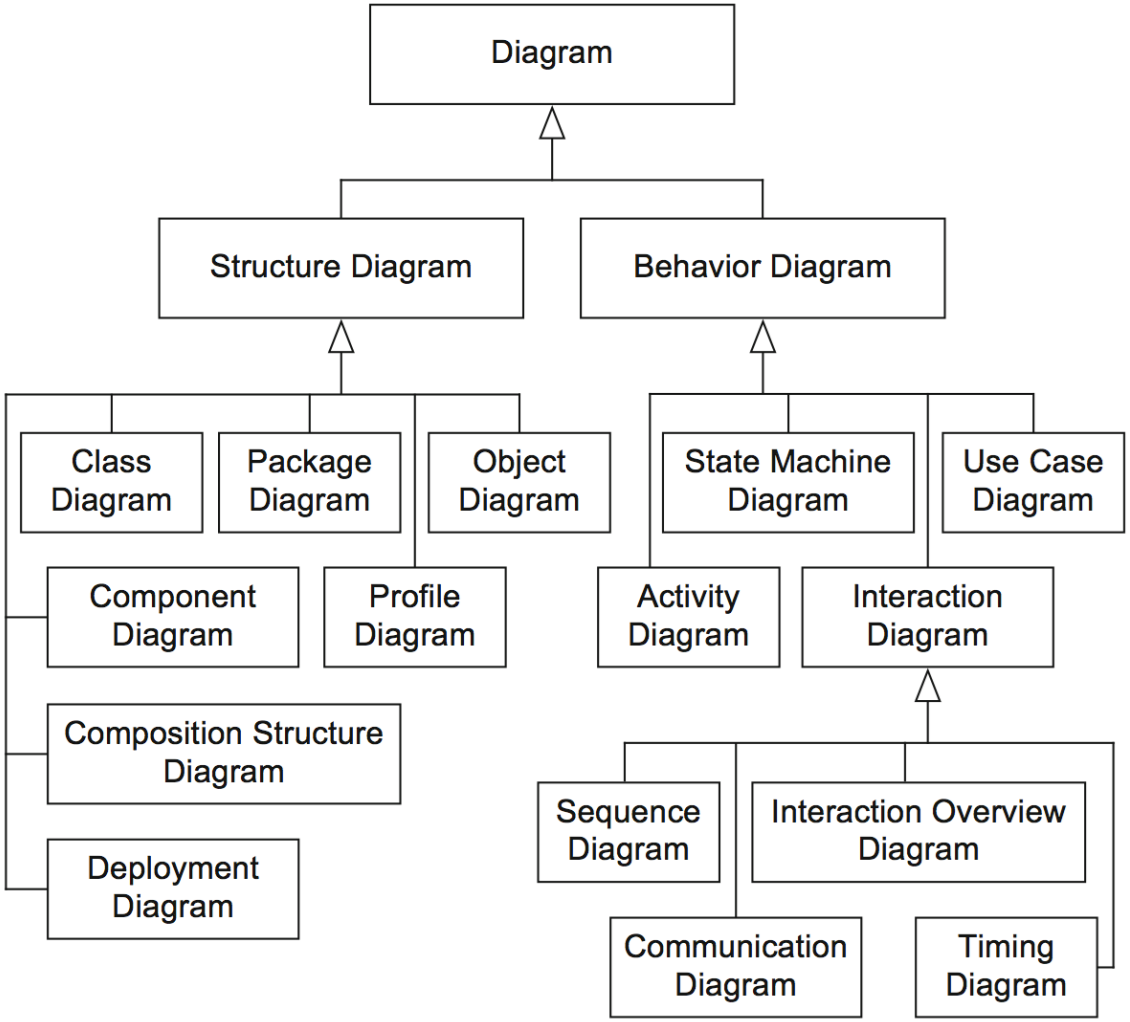
No additional attributes.

**Associations**

- subject : Classifier — References the subjects to which this use case applies. The subject or its parts realize all the use cases that apply to this subject. Use cases need not be attached to any specific subject, however. The subject may, but need not, own the use cases that apply to it.
- include : Include — References the Include relationships owned by this use case. (Specializes *Classifier.feature* and *Namespace.ownedMember*.)
- extend : Extend — References the Extend relationships owned by this use case. (Specializes *Classifier.feature* and *Namespace.ownedMember*.)
- extensionPoint: ExtensionPoint — References the ExtensionPoints owned by the use case. (Specializes *Classifier.feature* and *Namespace.ownedMember*.)

# UML diagrams

A UML model is represented graphically through **diagrams**



# Two types of UML diagrams

- Structure
  - Emphasize the static description of the elements of the system being modeled
  - Structural elements may have an associated behavior, e.g., operations in a class diagram
- Behavior
  - Behavior = the direct consequences of an action of at least one object
  - Affects how the states of objects change over time
  - Can be specified through the actions of a single object (e.g., state machine) or result from interactions between multiple objects (e.g., activity diagram)

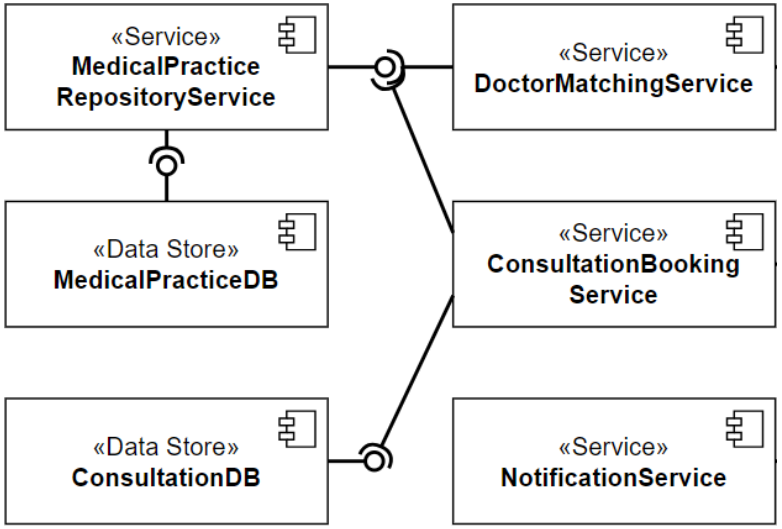
# Which diagrams will you use in this course?

- Class diagram
  - To define fine-grained structures within the system
- Package diagram
  - To describe the organization of coarse-grained implementation units
- Activity diagram
  - To specify system behavior via the control and data flow of actions

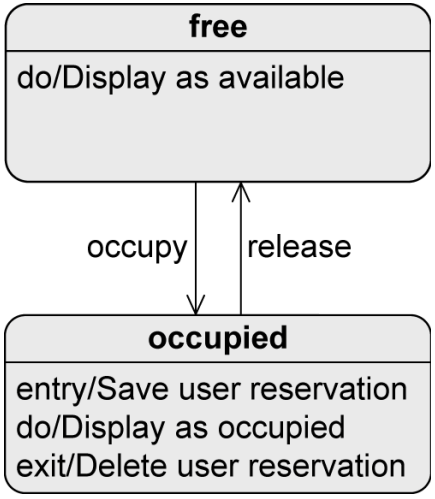
→Allows both structural and behavioral modeling to cover the overwhelming majority of any kind of system

# Which other UML diagrams can be useful today? (1/2)

- Component diagram
  For distributed systems with interprocess communication between components with clearly defined interfaces, e.g., service- or microservice-based systems

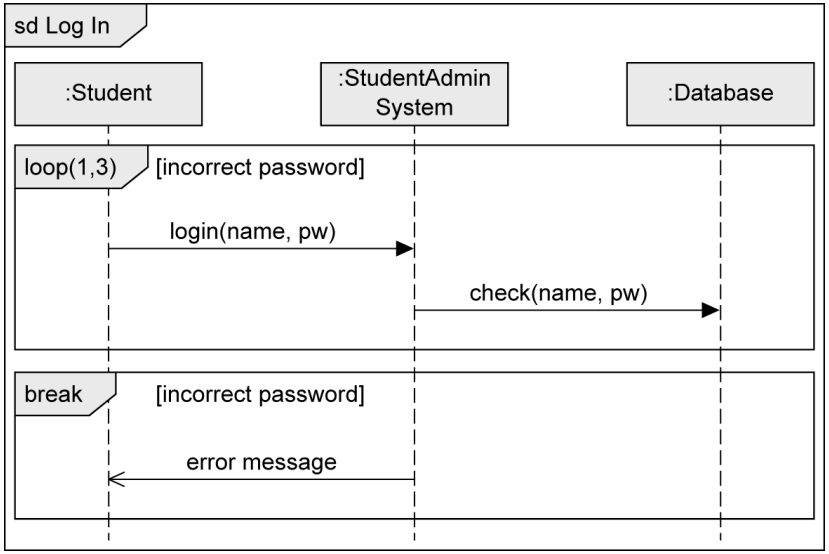



- State machine diagram
  For systems with a decent number of clearly defined states, between which the system transitions, e.g., cyber-physical systems like smart homes or robots

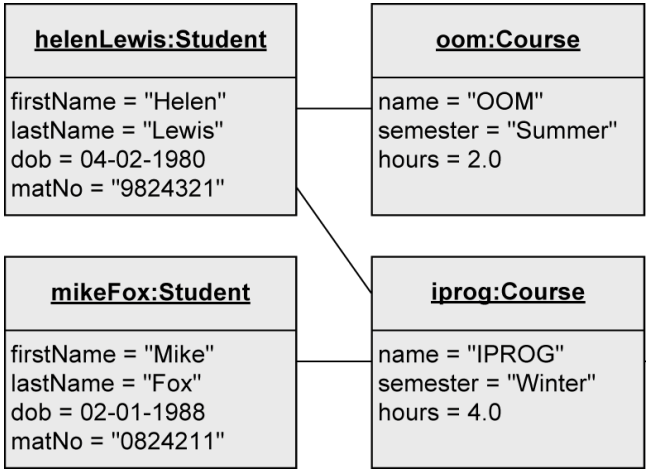


# Which other UML diagrams can be useful today? (2/2)

- Sequence diagram
  For systems with very complex, low-level method invocation flows that need to be modeled (activity diagrams are often easier to understand, but more suited to high-level flows)

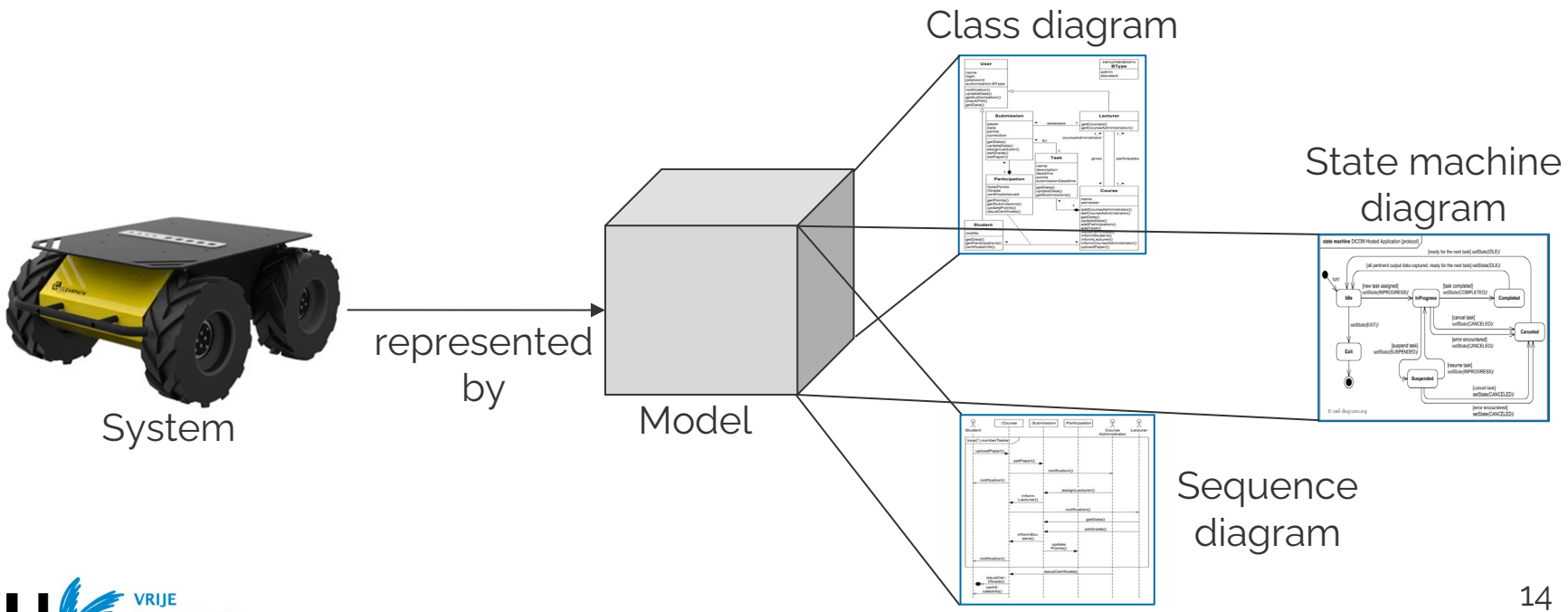



- Object diagram
  Situationally useful for reasoning about or debugging the creation of objects according to modeled constraints



# Models != diagrams

- A UML model contains everything related to your system
  - It is complete
- Diagrams are just “windows” on your model
  - Can be considered as projections of the same model
  - One diagram will show some parts of your model but not necessarily everything (abstraction!)



# What do its creators have to say about UML today? [1]

- Ivar Jacobson
  - "If UML is used in a smart way, it is very practical and results in products with higher value."
  - "UML has become complex and clumsy. For 80% of all software only 20% of UML is needed."
- Grady Booch
  - "IMHO the UML standard went off the rails when it was made overly complex to support MDD... the UML is not a prog lang"
  - "I still use it in my work"
- Bran Selic
  - "Although I do think that UML is bloated needlessly, I think it is mere pittance compared to what you find in the so-called "mainstream" languages"
  - "UML is the worst modeling language except for all the others"

[1] Jordi Cabot, "What do their creators think about UML now?", 2018, https://modeling-languages.com/uml-opinions-creators/