COMP 433 Software Engineering

Module 5: *System Modeling with UML*

Ahmed Tamrawi

atamrawi atamrawi.github.io ahmedtamrawi@gmail.com

UML

- The Unified Modeling Language (UML) is a consolidation of the best practices that have been established over the years in the use of modeling languages.
- UML enables us to present the *widely varying aspects of a software system* (e.g., requirements, data structures, data flows, and information flows) within a single framework using object-oriented concepts.

UML Usage

- UML is **not** tied to a specific development tool, specific programming language, or specific target platform on which the system to be developed must be used. Neither does UML offer a software development process.
- UML in fact separates the *modeling language* and *modeling method*.
- However, the language concepts of UML do favor an *iterative and incremental process*.

UML Usage

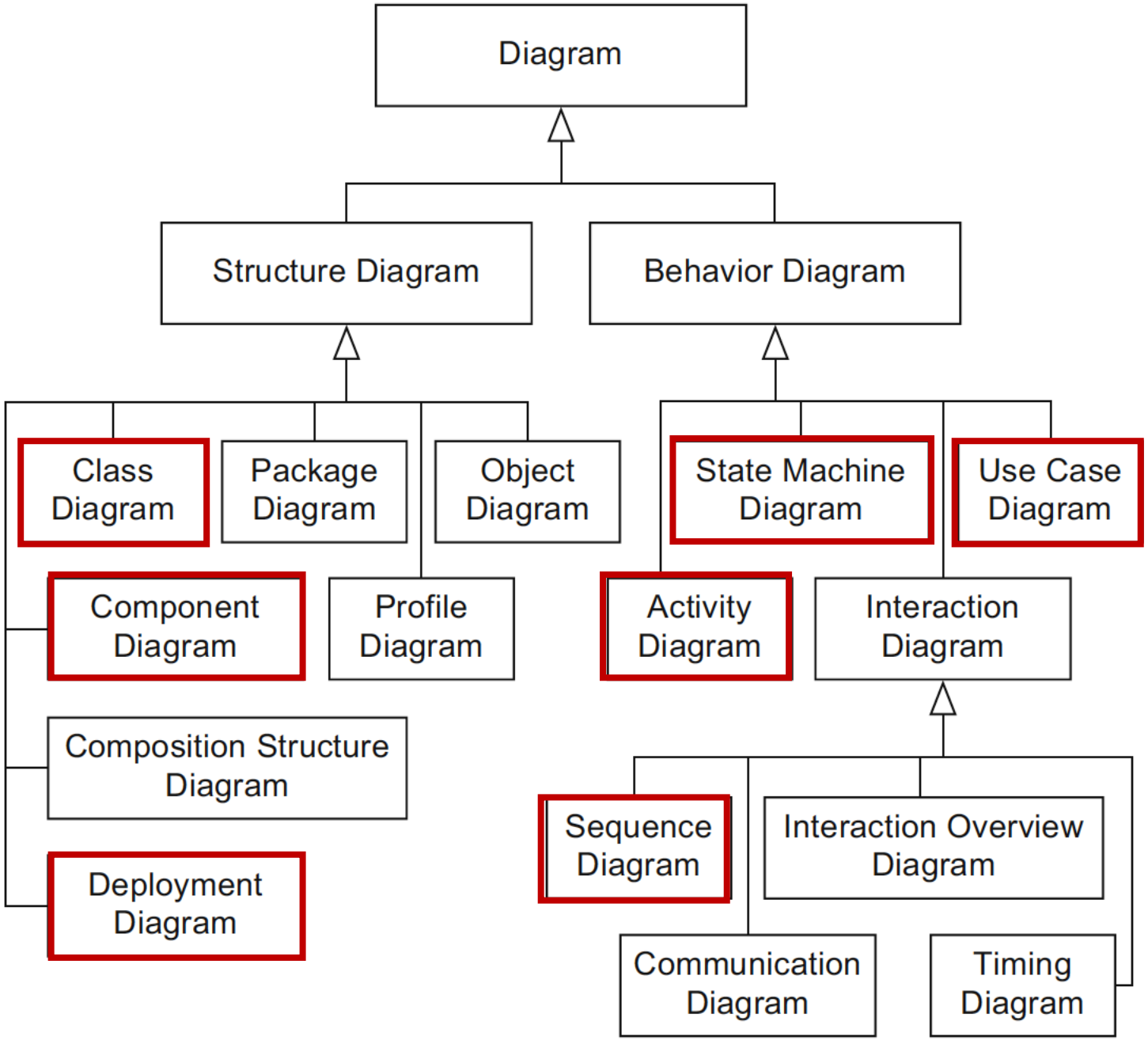
- UML can be used **consistently** across the entire software development process.
- At all stages of development, the same language concepts can be used in the same notation. Thus, a model can be refined in stages.
- There is no need for a model to be translated into another modeling language. This enables an iterative and incremental software development process.
- UML is well-suited for various application areas with different requirements regarding complexity, data volume, real time, etc.

UML Usage

- The UML model elements and their correct use are specified in the UML metamodel (https://www.omg.org/spec/UML/).
- The language concepts are defined so **generically** that a wide and flexible applicability is achieved.
- To avoid restricting the use of UML, the standard is (intentionally) vague at various points, permitting different interpretations in the form of semantic variation points.
- However, this is a two-edged sword; it also leads to *different implementations of the language standard* by modeling tools, which in turn, unfortunately makes it difficult to exchange models.

UML Diagrams

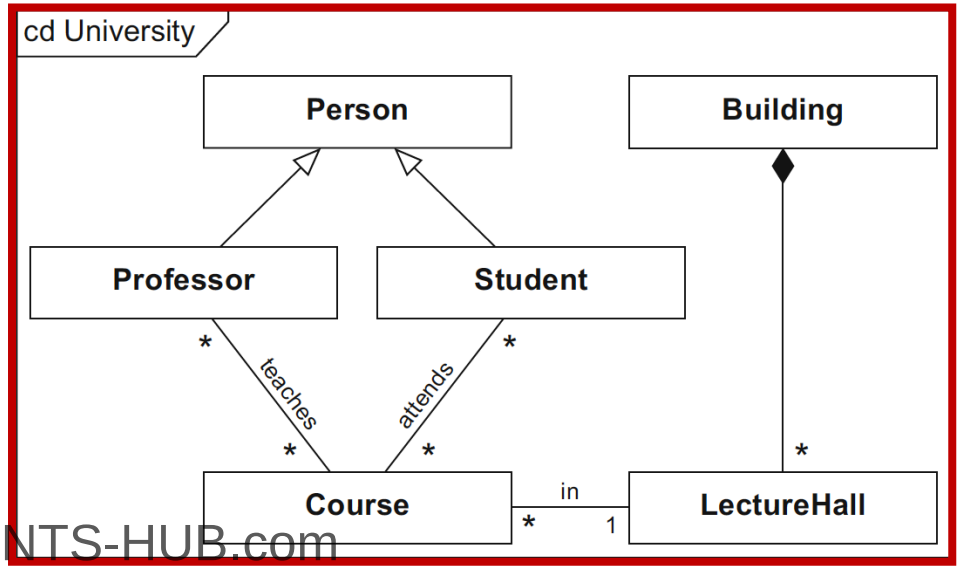
- In UML, a model is represented graphically in the form of diagrams which provides a *view of that part of reality* described by the model.
- There are diagrams that express *which users use which functionality* and diagrams that show the *structure of the system* but without specifying a concrete implementation.
- In the version 2.4.1, UML offers 14 diagrams that describe either the structure or the behavior of a system



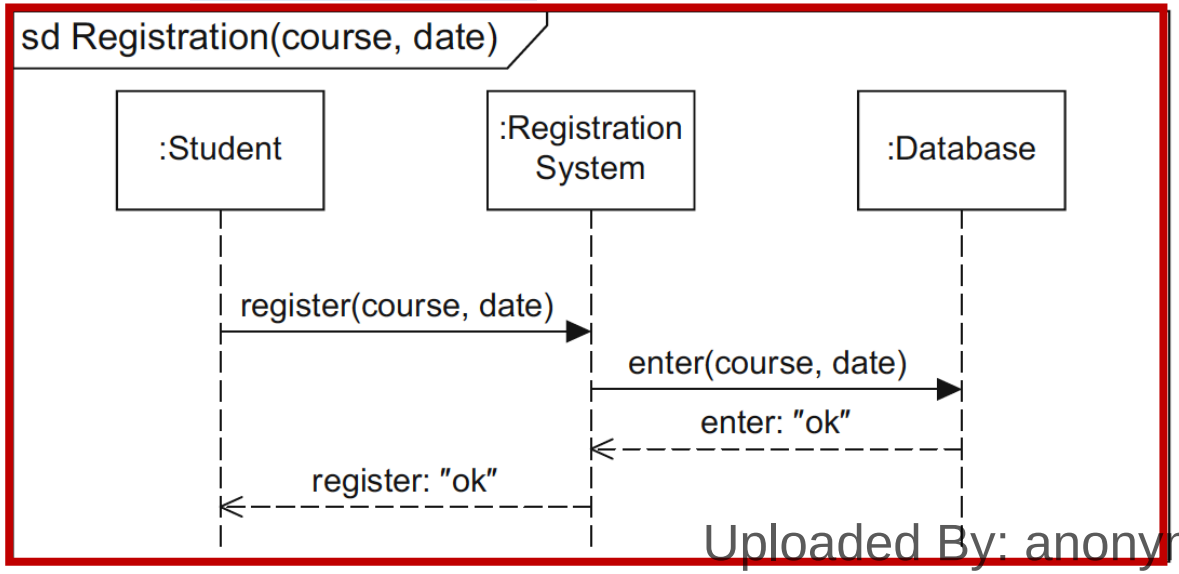
UML Diagrams

- A diagram is usually enclosed by a rectangle with a pentagon in the top left-hand corner. This pentagon contains the **diagram type** and the **frame name** of the diagram.
- Optionally, **parameters** may be specified following the name which then can be used within the diagram.

Class Diagram

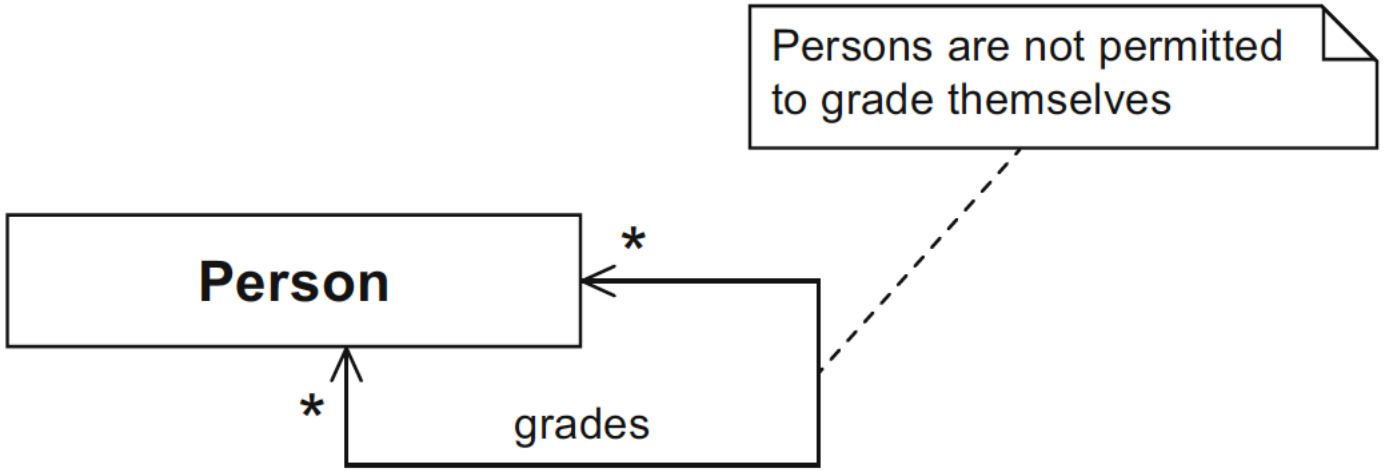


Sequence Diagram



UML Diagrams

- A concept that may occur in all diagrams is the note.
- A note can contain any form of expression that specifies the diagram and its elements more precisely—for example, in natural language or in the Object Constraint Language (OCL). Notes may be attached to all other model elements.

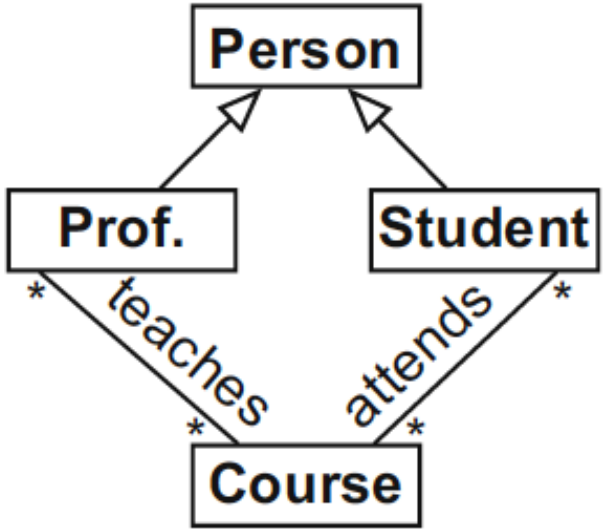


Structure Diagrams: *Class Diagram*

- The concepts of the class diagram originate from conceptual data modeling and object-oriented software development.
- These concepts are used to specify the *data structures* and *object structures* of a system.
- The class diagram is based primarily on the *concepts of class*, *generalization*, and *association*.

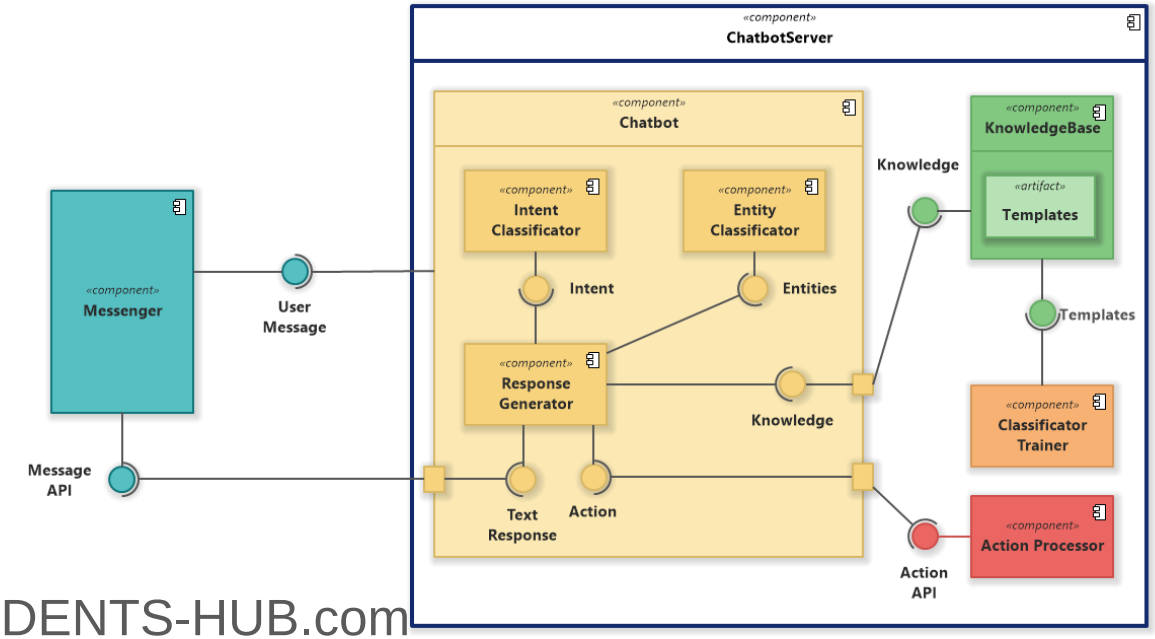
Structure Diagrams: *Class Diagram*

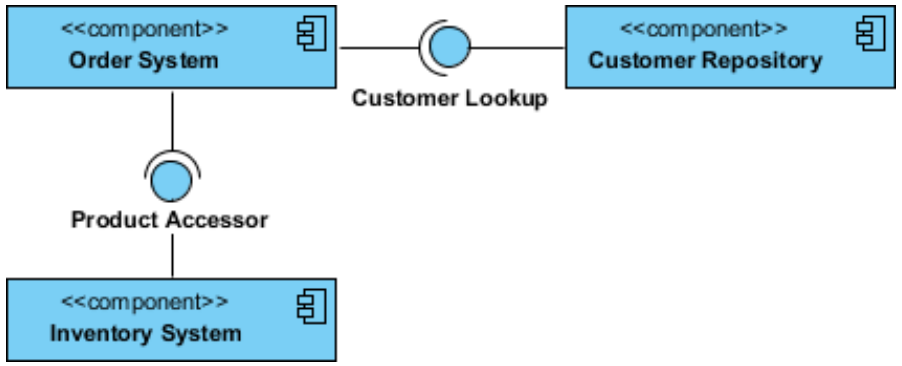
- For example, in a class diagram, you can model that the classes `Course`, `Student`, and `Professor` occur in a system.
  - `Professors` teach `Courses` and `Students` attend `Courses`.
  - `Students` and `Professors` have common properties as they are both members of the class `Person`. *This is expressed by a generalization relationship.*



Structure Diagrams: *Component Diagram*

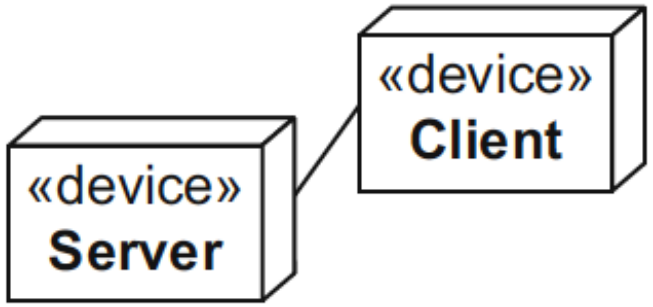
- UML pays homage to component-oriented software development by offering component diagrams.
- A component is an **independent**, **executable unit** that *provides other components with services* or *uses the services of other components*.



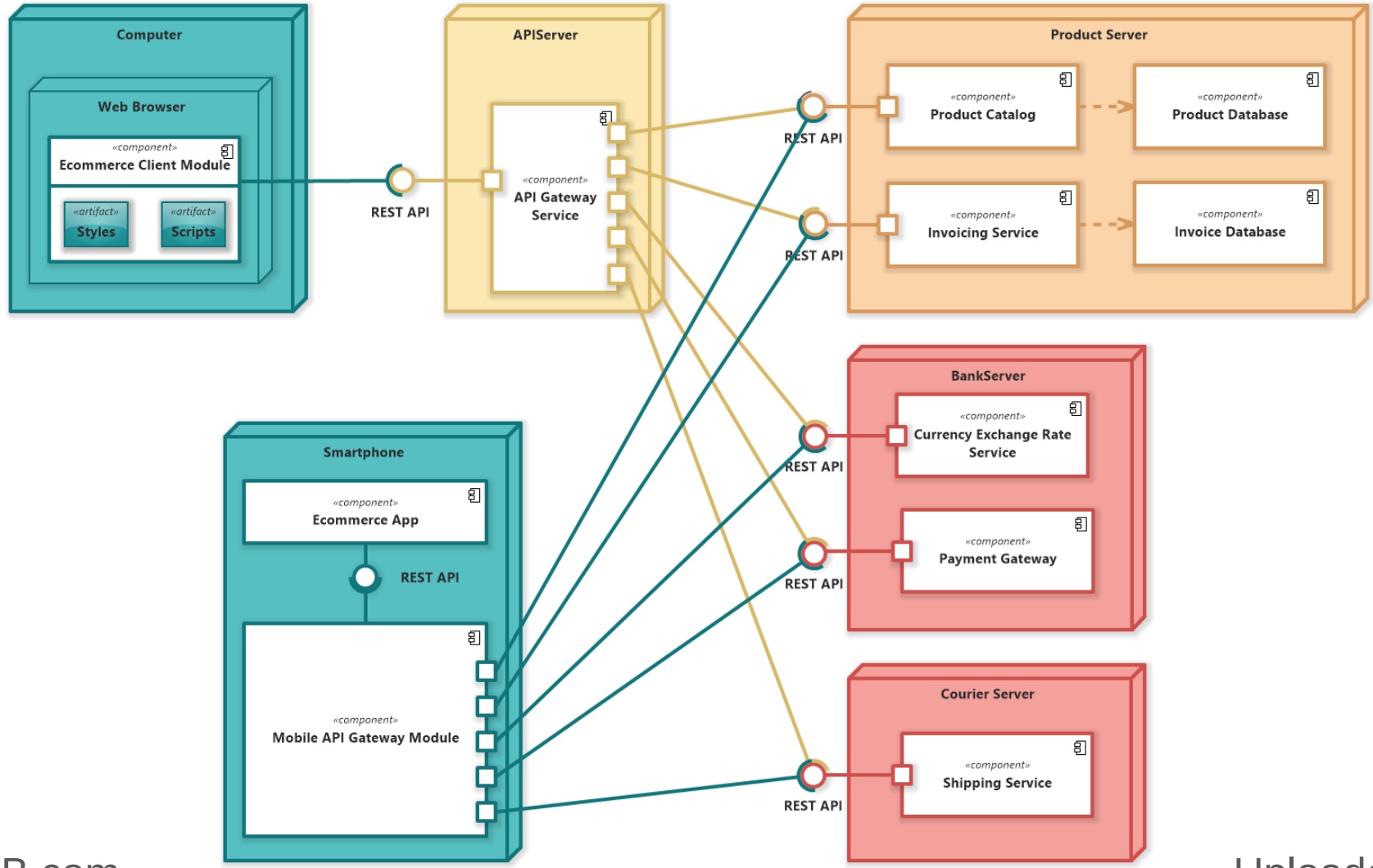


Structure Diagrams: *Deployment Diagram*

- The hardware topology used, and the runtime system assigned can be represented by the deployment diagram.
- The hardware encompasses processing units in the form of **nodes** as well as communication relationships between the **nodes**.
- A runtime system contains **artifacts** that are deployed to the **nodes**.

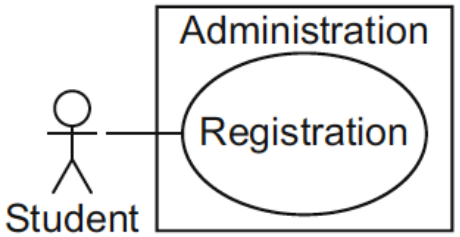


E-commerce Microservices Deployment Diagram



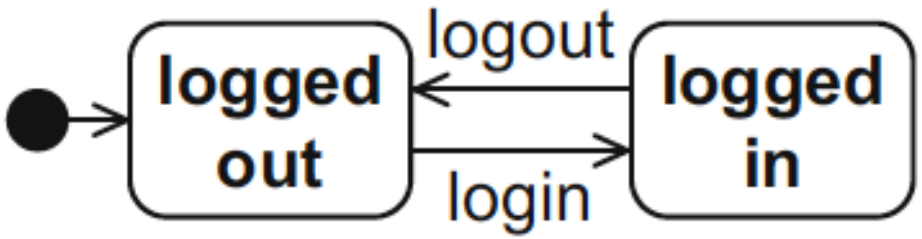
Behavior Diagrams: *Use Case Diagram*

- UML offers the use case diagram to enable you to *define the requirements that a system must fulfill*.
- This diagram describes which users use which functionalities of the system but does not address specific details of the implementation.
- The **units of functionality** that the system provides for its users are called use cases.
- In a university administration system, for example, the functionality **Registration** would be a use case used by **students**.



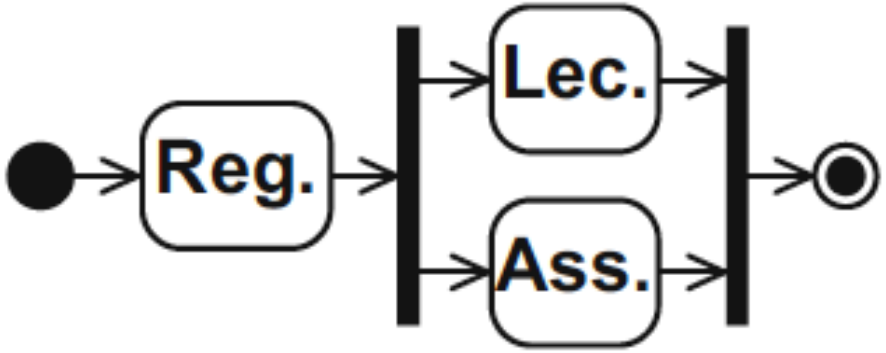
Behavior Diagrams: *State Machine Diagram*

- Within their life cycle, objects go through different states.
- For example, a person is in the state **logged out** when first visiting a website. The state changes to **logged in** after the person successfully entered username and password (*event* login). As soon as the person logs out (*event* logout), the person returns to the state **logged out**.
- This behavior can be represented in UML using the *state machine diagram* which describes the **permissible behavior** of an object in the form of possible states and state transitions triggered by various events.



Behavior Diagrams: *Activity Diagram*

- You can model processes of any kind using activity diagrams: both business processes and software processes.
- For example, an activity diagram can show which actions are necessary for a student to participate in a lecture and an assignment.
- Activity diagrams offer control flow mechanisms as well as data flow mechanisms that coordinate the actions that make up an activity, that is, a process.



Behavior Diagrams: *Sequence Diagram*

- The sequence diagram describes the *interactions between objects to fulfill a specific task*.
- The focus is on the *chronological order* of the messages exchanged between the **interaction partners**.
- Various constructs for controlling the chronological order of the messages as well as concepts for modularization allow you to model complex interactions.
- For example, registration for an exam in a university administration system.