UML-Diagram: Basics

Unified Modeling Language (UML) is a language for specifying, visualizing, constructing, and documenting the artifacts of software systems, as well as for business modeling and other non-software systems. It has vocabulary and rules to represent structural (static) and behavioral (dynamic) aspects of software systems. These aspects are used to specify precisely and completely the system structure and behavior.

Unified Modelling Language (UML) is a diagrammatic language/notation, used to model the system from different points of view. UML is basically a set of Graphical/Visual representations, called UML Diagrams. Different diagrams for different phases of Software Development, thus provide means to specify and document the design of a software system.

In UML, a model is represented graphically in the form of diagrams. A diagram provides a view of that part of reality described by the model, e.g., diagrams to express which users use which functionality and diagrams to show the structure of the system.

In architecture a building can be modeled from several views (or perspectives) such as ventilation perspective, electrical perspective, lighting perspective, heating perspective, etc. Just like architecture, UML facilitate to look at the architecture of a software system from different perspectives or views listed below:

- User's View
- Structural View
- Behavioral View
- Implementation View
- Environmental View

Normally UML's different diagrams are used to capture five different views of a system. In the following paragraphs we describe different views and specify UML diagram used for the view.

User's View: Captures the external users' view of the system in terms of the functionalities offered by the system. The users' view is a black box view of the system, where how the functionality is being provided is hidden from the user. In this view user's part of interaction with the system is depicted. The users' view can be considered as the central view and all other views are expected to conform to this view. It describes the external behavior of the system and is captured in the use case model. For use case model *Use Case Diagram* is used.

Structural View: In this view structure (static) of the system is depicted. Structural/Design view (logical view) describes the logical structures required to provide the functionality specified in the use's view (Use Case model). In this view Objects/Classes important to the understanding of the working of a system are identified. The structural view is captured in the *Domain model.* In domain model classes/objects and the relationships among the classes/ objects is depicted using *Class/Object Diagram.*

Behavioral View: Captures how different components of the system interact with each other to realize the system behavior. It is dynamic and time-dependent and describes concurrency in the system. OO techniques view software systems as systems of communicating objects and system components are represented in object/class form, so to depict behavioral view following UML diagrams are used:

- Activity Diagram
- Sequence Diagram
- Communication Diagram
- State Machine Diagram

Implementation View: Captures important components of the system and their dependencies. In this view internal workflow of the software is defined and it describes the physical software components of the system, such as executable files, class libraries and databases. To depict implementation view *Component Diagram* is used.

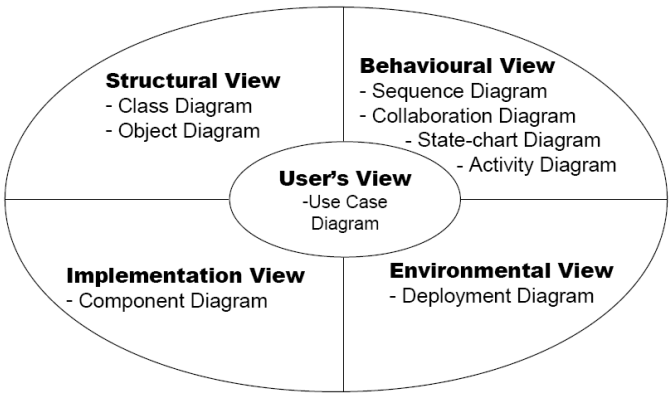
Environmental View: Captures how different components are implemented on different pieces of hardware. It explains after deployment behavior of the software model. It depicts where software components are physically installed on the hardware elements such as clients, servers, printers and the way they are connected.
To depict environmental view *Deployment Diagram* is used.

Not all the views will be required for every system. For instance, If your system runs on a single machine, you will not need the deployment view or the component view, as all the software components will be installed on one machine. For a simple system, one may have use case model, domain model (basic class diagram) and one of the interaction diagrams.

Five different ways of viewing a system is supported in UML by modelling techniques to capture each view. Just as a photograph, family tree and description give us different views of a person, the UML views show us a system from different points of view.

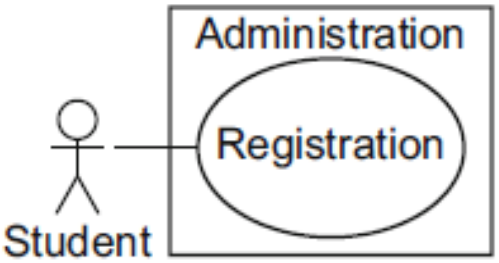
Each one offers a different perspective, no one gives us the whole picture. To gain complete understanding of the system all the views must be considered.

Five views and UML diagrams used for different views is depicted in the following diagram:

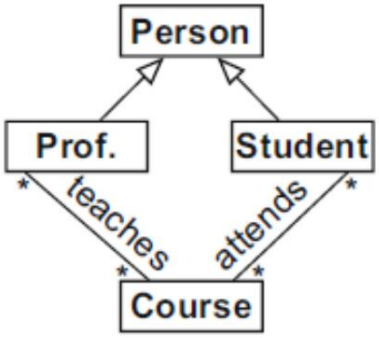


Now we briefly describe UML Diagrams, syntax and rules of each diagram will be described when we will create those diagrams during analysis and design phases.

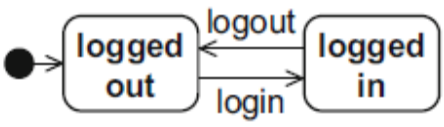
Use Case Diagram: Depicts external interaction with actors and it shows how the system interacts with its users. Use Case is represented by ellipse with use case name inside it. Actors are represented by a stick symbol; a rectangle shows the boundary of the system and name of the system may be mentioned inside the boundary. Solid line indicates association between actor and use cases. For example, in a university administration system, the functionality registration would be a use case used by students as shown below. In the diagram *Student* is an actor, *Registration* is a use case.



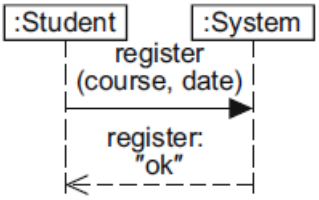
Class/Object Diagram: Captures static structural aspects, objects/classes and their relationships. It also captures data elements in the system and the relationships between them. For example, in a class diagram, you can model that the classes Course, Student, and Professor occur in a system. Professors teach courses and students attend courses. Students and professors have common properties as they are both members of the class Person as shown below:



State Machine Diagram: Depicts how the different objects of a single class behave through all the use cases in which the class is involved. It depicts dynamic state behavior. In the example diagram shown below 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.

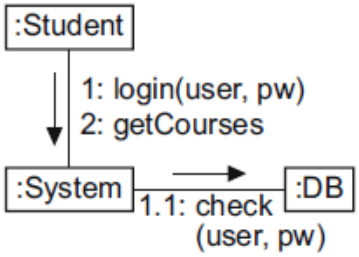


Sequence Diagram: Models object interaction over time to achieve the functionality of a use case. The sequence diagram depicts the interactions between objects to fulfill a specific task. For example registration for an exam in a university administration system. The focus is on the chronological order of the messages exchanged between the interaction partners as shown below:

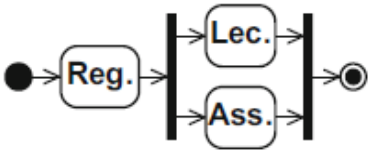


In sequence diagram various constructs for controlling the chronological order of the messages as well as concepts for modularization allow you to model complex interactions.

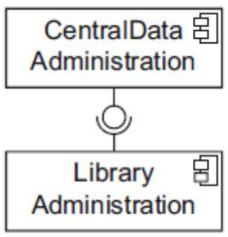
Communication Diagram: Models' component interaction and structural dependencies. It illustrates the communication between different objects. Communication diagram clearly shows who interacts with whom. Focus is on the communication relationships between the interaction partners as shown below:



Activity Diagram: Models object activities. The sequence of activities that make up a process is illustrated. For example, an activity diagram shown below depicts which actions are necessary for a student to participate in a lecture and an assignment.



Component Diagram: Models software architecture. Depict different software components of the system and the dependencies between them. A component is an independent, executable unit that provides other components with services or uses the services of other components as shown below:



Deployment Diagram: Models physical architecture. Depict software and hardware elements of the system and the physical relationships between them. Deployment diagram represents hardware topology used and the runtime system assigned as shown in the following 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.