Software Development Using UML: Basics

Introduction

In Object Oriented software development, the whole development process is based round the Object - a thing or concept that initially represents something in problem domain of a real world and eventually ends up as a component of the system code. An object-oriented system is made up of objects that collaborate to achieve the required functionality, what the system being developed has to provide.

The Unified Modelling Language (UML) is a set of diagrammatic techniques, which are specifically tailored for object-oriented development.

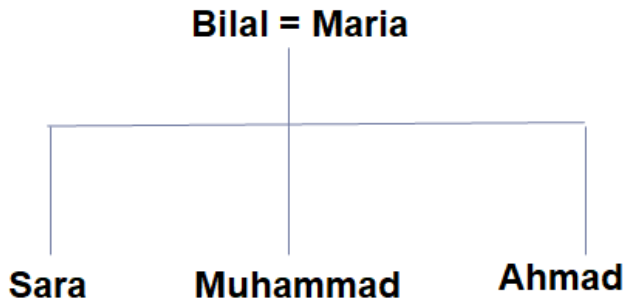
UML is simply a notation or language for development; it is not in itself a development method and does not include detailed instructions on how it should be used in a project.

Modelling

Architects and engineers have always used special types of drawing to help them to describe what they are designing and building. Software developers also use specialized diagrams to model the system that they are working on throughout the development process. Each model produced represents part of the system or some aspect of it, such as the structure of the stored data, or the way operations are carried out. Each model provides a view of the system, but not the whole picture.

For example, models in architecture, e.g., construction plans. A construction plan for a building contains information such as the floor plan. Construction materials to be used are not specified at this point in time; they are irrelevant and would make the plan more complicated than necessary. The plan also does not contain any information about how the electrical cables are to be laid. A separate plan is created for this aspect to avoid presenting too much information at once.
Just like in architecture, it is important in information technology that people with different backgrounds can read, interpret, and implement the model

As an example, let us consider a small boy, Ahmad, and imagine that we have his photograph. The photograph is one possible model of the real-life Ahmad; it tells us what he looks like and may give some idea of his character. Listed below is another model of real-life Ahmad.



The family tree (a graphical, diagrammatic model) in above figure tells us nothing about Ahmad's appearance, but it does provide information about his family. We can see that he has one sister, one brother and he is the youngest child; we also find the first names of Ahamad's parents and siblings.

Ahmad Bilal

Date of Birth:	May 6, 2010
Address:	House 98, Street 50, G-14/2, Islamabad
School:	SuperNova School
Class:	5A

Above stated information, is a textual model; it tells us some details about Ahmad, his birth date, address, school, and which class he is in.

Each model (the photograph, the family tree and the piece of text) tells us something about Ahmad, but we can only get all the information by looking at all the models (in fact we would need many more detailed models to get a complete picture of his).

Models allow us to describe systems efficiently and elegantly. A system is an integrated whole made up of components that are related to one another and influence each other in such a way that they can be perceived as a single, task-based or purpose-based unit. In this regard, they separate themselves from the surrounding environment. Examples of systems are material things, such as cars or airplanes, ecological environments, such as lakes and forests, but also organizational units such as a university or a company. In information technology, we are interested in software systems and thus in models that describe software systems.

Abstraction

The characteristic of a model to provide some but not all the information about the person or thing being modelled is known as abstraction. We can say that each of the models depicted above is an abstraction of the real-life Ahamad, focusing on certain aspects of his and ignoring other details. In the same way, each of the modelling techniques in the Unified Modelling Language provides a particular view of the system as it develops; each UML model is an abstraction of the complete system.

Abstraction is a very important tool in modelling any sort of software system because typically the problems developers have to deal with are much more complex for one developer to hold all the details in his head at once. Abstraction provides a means of concentrating on only those aspects of the system that are currently of interest and putting other details to the side for the time being.

Abstraction means generalization - setting aside specific and individual features. Abstract is the opposite of concrete. Abstracting therefore means moving away from specifics, distinguishing the substance from the incidental, recognizing common characteristics.

Object Orientation

The object-oriented approach corresponds to the way we look at the real world; we see it as a society of autonomous individuals, referred to as objects, which take a fixed place in this society and must thereby fulfill predefined obligations.

Viewed simply, objects are elements in a system whose data and operations are described. Objects interact and communicate with one another.

Classes describe the attributes and the behavior of a set of objects (the instances of a class) abstractly and thus group common features of objects. For example, people have a name, an address, and NIC number. Courses have a unique identifier, title and a description. Lecture halls have a name as well as a location, etc. A class also defines a set of permitted operations that can be applied to the instances of the class. For example, you can reserve a lecture hall for a certain date, a student can register for an exam, etc. In this way, classes describe the behavior of objects. Classes are 'frames' for objects (basically the 'types' for the objects).

Object the instances of a class. For example, 217 the main Lecture Hall of CS-dept. of QAU, is a concrete instance of the class LectureHall. In particular an object is distinguished by the fact that it has its own identity, that is, different instances of a class can be uniquely identified. An object always has a certain state. A state is expressed by the values of its attributes. For example, a lecture room can have the state occupied or free. An object also displays behavior. The behavior of an object is described by the set of its operations. Operations are triggered by sending a message. Objects communicate with one another through messages.

A ***message*** to an object represents a request to execute an operation. The object itself decides whether and how to execute this operation. The operation is only executed if the sender is authorized to call the operation.

The concept of ***inheritance*** is a mechanism for deriving new classes from existing classes. A subclass derived from an existing class (referred as superclass) inherits all visible attributes and operations (specification and implementation) of the superclass.
A subclass can:

- Define new attributes and/or operations.
- Overwrite the implementation of inherited operations.
- Add its own code to inherited operations.

A mechanism for expressing similarity between things thus simplifying their definition. Inheritance enables extensible classes and as a consequence, the creation of class hierarchies as the basis for object-oriented system development.
A class hierarchy consists of classes with similar properties. For example,

Person ← Employee ← Faculty
Person ← Student where X← Y means that Y is a subclass of X.

Encapsulation is combination of attributes and operations into a single entity. A technique in which data are packaged together (Object) with their corresponding procedures. The interface to each object is defined in such a way as to reveal as little as possible about its inner workings thus referred as information hiding. Encapsulation is the protection against unauthorized access to the internal state of an object via a uniquely defined interface.

Generally, ***polymorphism*** is the ability to adopt different forms. During the execution of a program, a polymorphic attribute can have references to objects from different classes. When this attribute is declared, a type (e.g., class Person) is assigned statically at compile time. At runtime, this attribute can also be bound dynamically to a subtype (e.g., subclass Employee or subclass Student). The ability to hide different implementations behind a common interface. The ability for two or more objects to respond to the same request, each in its own way. A polymorphic operation can be executed on objects from different classes and have different semantics in each case. For example, H_2O is water and can be responded by different objects as liquid(water), ice(solid), and steam(vapor) depending on the situation the object is representing.

UML: Basics

UML is a diagrammatic language or notation, providing a set of diagramming techniques that model the system from different points of view. 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. 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. UML diagrams describe either the structure or the behavior of a system.

The principal UML diagrams with brief descriptions:

Use Case *Diagram:* How the system interacts with its users. External interaction with actors.

Class/Object Diagram: Captures static structural aspects, objects and relationships. The data elements in the system and the relationships between them.

Sequence Diagram: Models object interaction over time to achieve the functionality of a use case.

Collaboration Diagram: Models component interaction and structural dependencies.

State Diagram: How the different objects of a single class behave through all the use cases in which the class is involved. Depicts Dynamic state behavior.

Activity Diagram: The sequence of activities that make up a process. Models object activities.

Component Diagram: The different software components of the system and the dependencies between them. Models software architecture.

Deployment Diagram: The software and hardware elements of the system and the physical relationships between them. Models physical architecture.

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 following different perspectives or views:

1. Use Case view
2. Structural/Design view
3. Behavioral/Process view
4. Implementation view
5. Deployment view

The use case view of the system is the users' view: it specifies what the user wants the system to do; the other 4 views describe how to achieve this. The use case view describes the external behavior of the system and is captured in the use case model. Users' view captures the external users' view of the system in terms of the functionalities offered by the system. It means user's part of interaction with the system.
In the early stages of development, the software architecture is driven by the use cases; we model the system's software in terms of collaborations of the classes required by each use case.

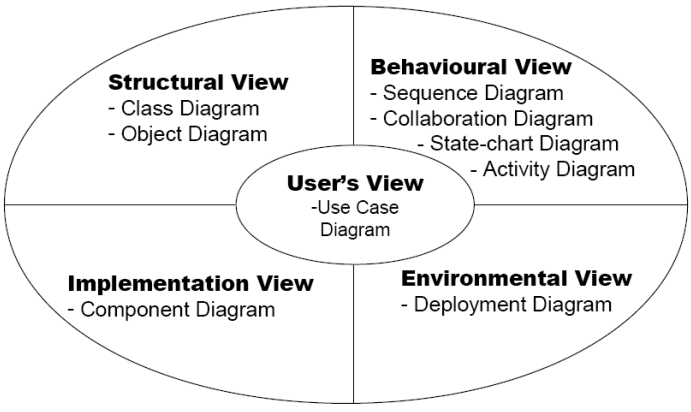
The structural/design view (sometimes called the logical view) describes the logical structures required to provide the functionality specified in the use case view. Structure of the model (static) is depicted. Captures the relationships among the classes. This view describes the classes (including attributes and operations) of the system and their interactions. This view is captured in the class diagram and the interaction diagrams.

Behavioral/Process view captures how classes/objects interact with each other to realize the system behavior (time-dependent/dynamic). This view is concerned with describing concurrency in the system, which is achieved through sequence diagrams, activity diagram, collaboration diagram and state chart diagram.

The implementation view captures important components of the system and their dependencies. Internal workflow of the software is defined. This view describes the physical software components of the system, such as executable files, class libraries and databases. This view of the system can be modelled using component diagrams.

The deployment view of the system describes the hardware components of the system such as clients, servers, printers and the way they are connected. This view can also be used to show where software components are physically installed on the hardware elements. This view explains after deployment behavior of the software model. Deployment diagrams describe this view of the system.

Not all the views will be required for every system. For instance, if your system does not use concurrency, you will not need a process view. 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.
Relationship among fives views and principal UML diagrams is depicted below:



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.

An important tool for modelling software systems is decomposition; breaking down of a large, complex problem or system into successively smaller parts, until each part is small chunk and can be worked on as an independent unit. In object oriented, systems are decomposed according to the data they have to store, access and manipulate. Initially, the models that are constructed using the techniques provided by the UML help the developer to impose a coherent structure on the information gathered from clients and users. Main use of the models provides a basis for discussions, since talking about the system as represented helps to identify gaps and inconsistencies in the information. As the project progresses, the original models are enhanced with details relating to design and implementation issues, so that eventually they become a blueprint for the development of the system.