Unified Modeling Language (UML)

Unified Modeling Language (UML) is a general purpose modelling language. The main aim of UML is to define a standard way to **visualize** the way a system has been designed. It is quite similar to blueprints used in other fields of engineering.
UML is **not a programming language**; it is rather a visual language. We use UML diagrams to portray the **behavior and structure** of a system. UML helps software engineers, businessmen and system architects with modeling, design and analysis. The Object Management Group (OMG) adopted Unified Modeling Language as a standard in 1997. Its been managed by OMG ever since. International Organization for Standardization (ISO) published UML as an approved standard in 2005.

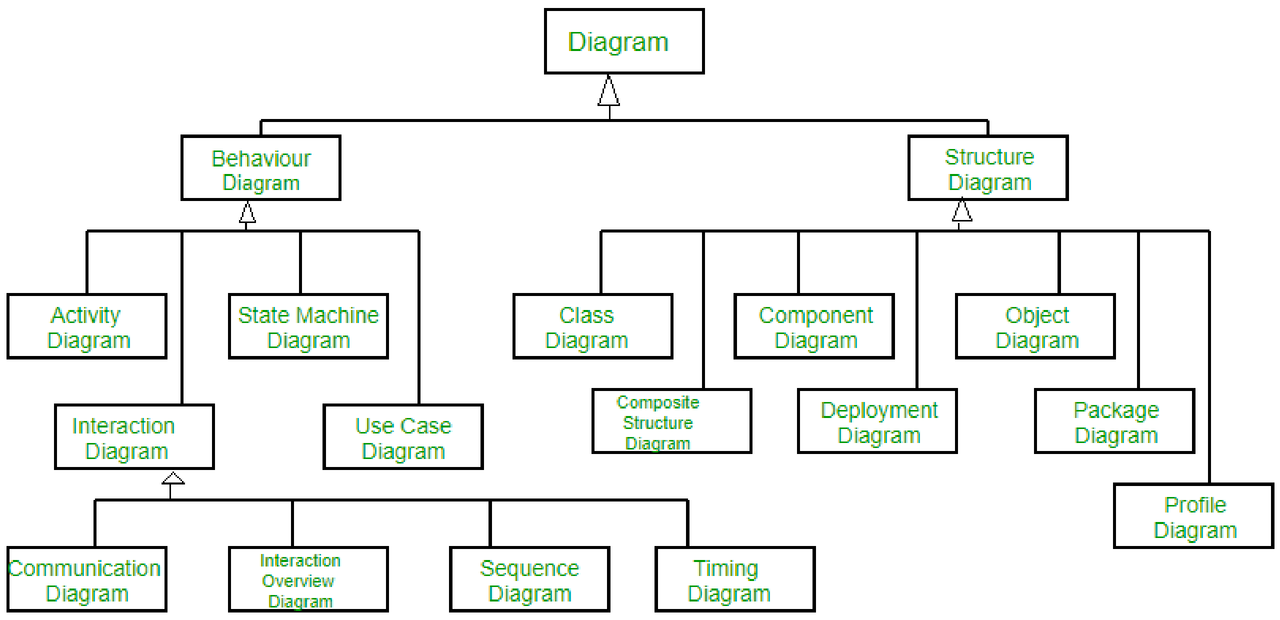
Do we really need UML?

- Complex applications need collaboration and planning from multiple teams and hence require a clear and concise way to communicate amongst them.
- Businessmen do not understand code. So UML becomes essential to communicate with non programmers essential requirements, functionalities and processes of the system.
- A lot of time is saved down the line when teams are able to visualize processes, user interactions and static structure of the system.

UML is linked with **object oriented** design and analysis. UML makes the use of elements and forms associations between them to form diagrams. Diagrams in UML can be broadly classified as:

1. **Structural Diagrams –** Capture static aspects or structure of a system. Structural Diagrams include: Component Diagrams, Object Diagrams, Class Diagrams and Deployment Diagrams.
2. **Behavior Diagrams –** Capture dynamic aspects or behavior of the system. Behavior diagrams include: Use Case Diagrams, State Diagrams, Activity Diagrams and Interaction Diagrams.

The image below shows the hierarchy of diagrams according to UML 2.2



Object Oriented Concepts Used in UML –

1. **Class –** A class defines the blue print i.e. structure and functions of an object.
2. **Objects –** Objects help us to decompose large systems and help us to modularize our system. Modularity helps to divide our system into understandable components so that we can build our system piece by piece. An object is the fundamental unit (building block) of a system which is used to depict an entity.
3. **Inheritance –** Inheritance is a mechanism by which child classes inherit the properties of their parent classes.
4. **Abstraction –** Mechanism by which implementation details are hidden from user.
5. **Encapsulation –** Binding data and functions together and protecting it from the outer world is referred to as encapsulation.
6. **Polymorphism –** Mechanism by which functions or entities are able to exist in different forms.

Additions in UML 2.0 –

- Software developments methodologies like agile have been incorporated and scope of original UML specification has been broadened.

- Originally UML specified 9 diagrams. UML 2.x has increased the number of diagrams from 9 to 13. The four diagrams that were added are: timing diagram, communication diagram, interaction overview diagram and composite structure diagram. UML 2.x renamed statechart diagrams to state machine diagrams.
- UML 2.x added the ability to decompose software system into components and sub-components.

Structural UML Diagrams –

1. **Class Diagram –** The most widely use UML diagram is the class diagram. It is the building block of all object oriented software systems. We use class diagrams to depict the static structure of a system by showing system's classes, their methods and attributes. Class diagrams also help us identify relationship between different classes or objects.
2. **Composite Structure Diagram –** We use composite structure diagrams to represent the internal structure of a class and its interaction points with other parts of the system. A composite structure diagram represents relationship between parts and their configuration which determine how the classifier (class, a component, or a deployment node) behaves. They represent internal structure of a structured classifier making the use of parts, ports, and connectors. We can also model collaborations using composite structure diagrams. They are similar to class diagrams except they represent individual parts in detail as compared to the entire class.
3. **Object Diagram –** An Object Diagram can be referred to as a screenshot of the instances in a system and the relationship that exists between them. Since object diagrams depict behaviour when objects have been instantiated, we are able to study the behaviour of the system at a particular instant. An object diagram is similar to a class diagram except it shows the instances of classes in the system. We depict actual classifiers and their relationships making the use of class diagrams. On the other hand, an Object Diagram represents specific instances of classes and relationships between them at a point of time.

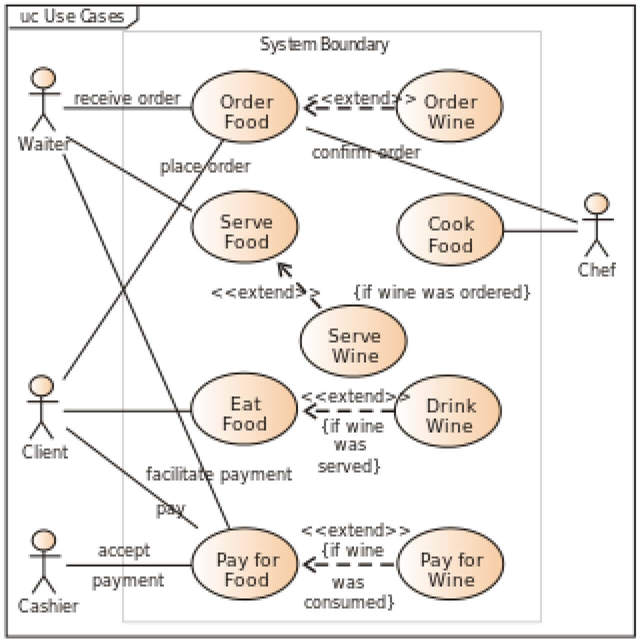
4. **Component Diagram –** Component diagrams are used to represent the how the physical components in a system have been organized. We use them for modeling implementation details. Component Diagrams depict the structural relationship between software system elements and help us in understanding if functional requirements have been covered by planned development. Component Diagrams become essential to use when we design and build complex systems. Interfaces are used by components of the system to communicate with each other.
5. **Deployment Diagram –** Deployment Diagrams are used to represent system hardware and its software. It tells us what hardware components exist and what software components run on them. We illustrate system architecture as distribution of software artifacts over distributed targets. An artifact is the information that is generated by system software. They are primarily used when software is being used, distributed or deployed over multiple machines with different configurations.
6. **Package Diagram –** We use Package Diagrams to depict how packages and their elements have been organized. A package diagram simply shows us the dependencies between different packages and internal composition of packages. Packages help us to organize UML diagrams into meaningful groups and make the diagram easy to understand. They are primarily used to organize class and use case diagrams.

Behavior Diagrams –

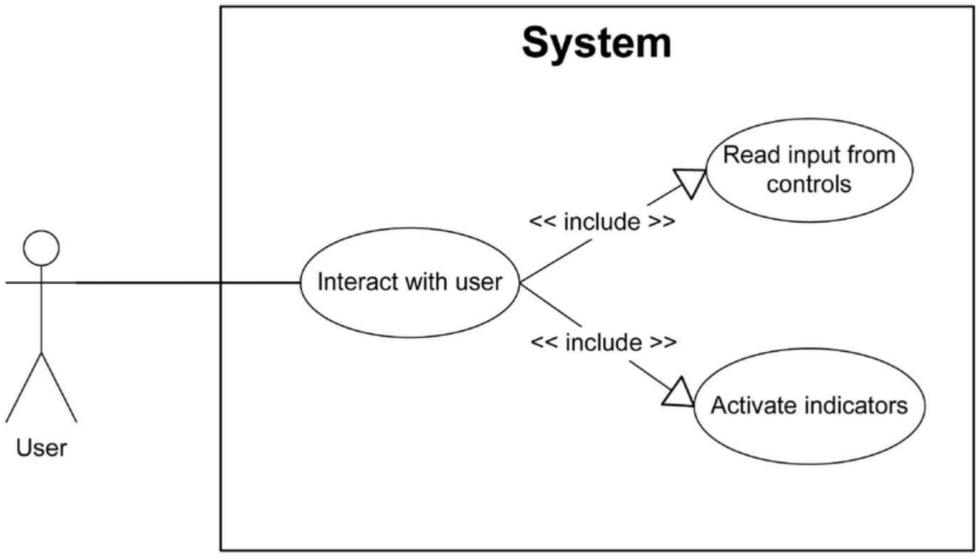
1. **State Machine Diagrams –** A state diagram is used to represent the condition of the system or part of the system at finite instances of time. It's a behavioral diagram and it represents the behavior using finite state transitions. State diagrams are also referred to as **State machines** and **State-chart Diagrams.** These terms are often used interchangeably. So simply, a state diagram is used to model the dynamic behavior of a class in response to time and changing external stimuli.
2. **Activity Diagrams –** We use Activity Diagrams to illustrate the flow of control in a system. We can also use an activity diagram to refer to the steps involved in the execution of a use case. We model

sequential and concurrent activities using activity diagrams. So, we basically depict workflows visually using an activity diagram.An activity diagram focuses on condition of flow and the sequence in which it happens. We describe or depict what causes a particular event using an activity diagram.

3. **Use Case Diagrams –** Use Case Diagrams are used to depict the functionality of a system or a part of a system. They are widely used to illustrate the functional requirements of the system and its interaction with external agents (actors). A use case is basically a diagram representing different scenarios where the system can be used. A use case diagram gives us a high level view of what the system or a part of the system does without going into implementation details.
4. **Sequence Diagram –** A sequence diagram simply depicts interaction between objects in a sequential order i.e. the order in which these interactions take place. We can also use the terms event diagrams or event scenarios to refer to a sequence diagram. Sequence diagrams describe how and in what order the objects in a system function. These diagrams are widely used by businessmen and software developers to document and understand requirements for new and existing systems.
5. **Communication Diagram –** A Communication Diagram (known as Collaboration Diagram in UML 1.x) is used to show sequenced messages exchanged between objects. A communication diagram focuses primarily on objects and their relationships. We can represent similar information using Sequence diagrams; however, communication diagrams represent objects and links in a free form.
6. **Timing Diagram –** Timing Diagram are a special form of Sequence diagrams which are used to depict the behavior of objects over a time frame. We use them to show time and duration constraints which govern changes in states and behavior of objects.
7. **Interaction Overview Diagram –** An Interaction Overview Diagram models a sequence of actions and helps us simplify complex interactions into simpler occurrences. It is a mixture of activity and sequence diagrams.



A business Use case diagram depicts a model of several *business use cases* (goals) which represents the interactions between a restaurant (the business system) and its primary stakeholders (*business actors* and *business workers*).



Example: Let us consider the purchasing process in user-interaction diagrams. purchase use case is a frame that includes (uses) three other frames (use cases): query catalogue, shopping cart, and confirm proceed. These use cases are described by means of separate user interaction diagrams. In addition, the logic of the use cases is not modified in the purchase user interaction diagram. It integrates the logic query catalogue diagram by checking which buttons (i.e., exit) the user pressed when he or she exits from shopping cart. The developer can also identify an inclusion relationship between manage catalogue and withdraw article, Modify article and add article use cases. In these cases, four windows can be optionally opened (depending on a menu) from the manage catalogue window. In addition, the administrator identification window is mandatory opened from the Manage catalogue window in order to achieve the system's administrator tasks.

Figure A user-interface diagram for the purchasing process

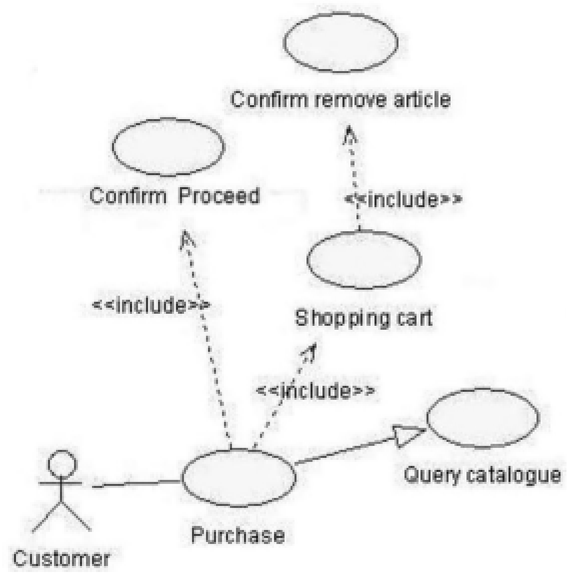


Figure A piece of the user interface diagram of the administrator side

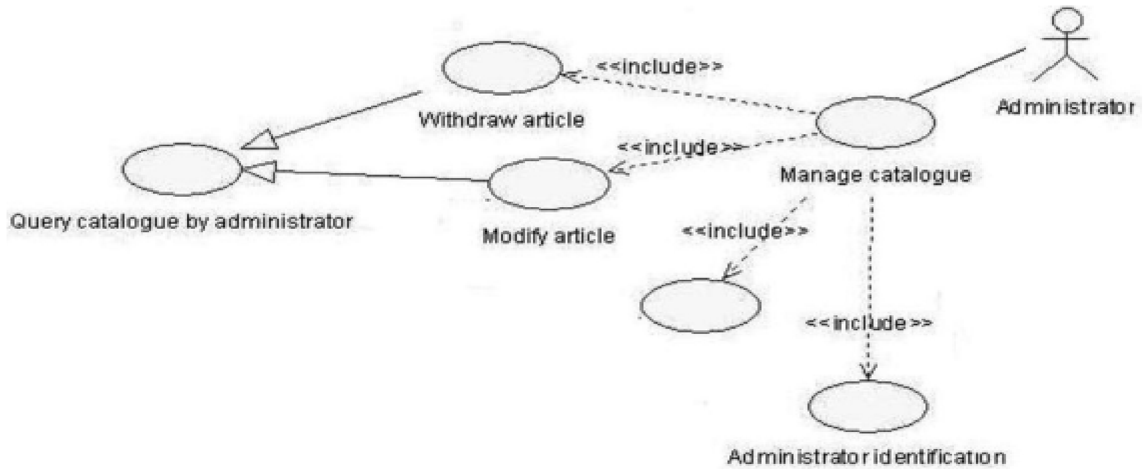
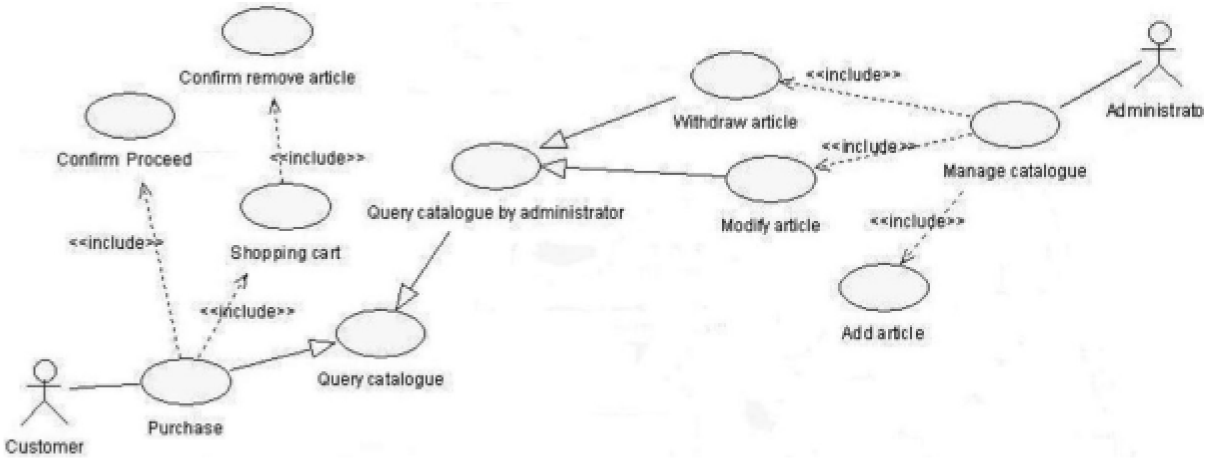


Figure . A piece of the user-interface diagram



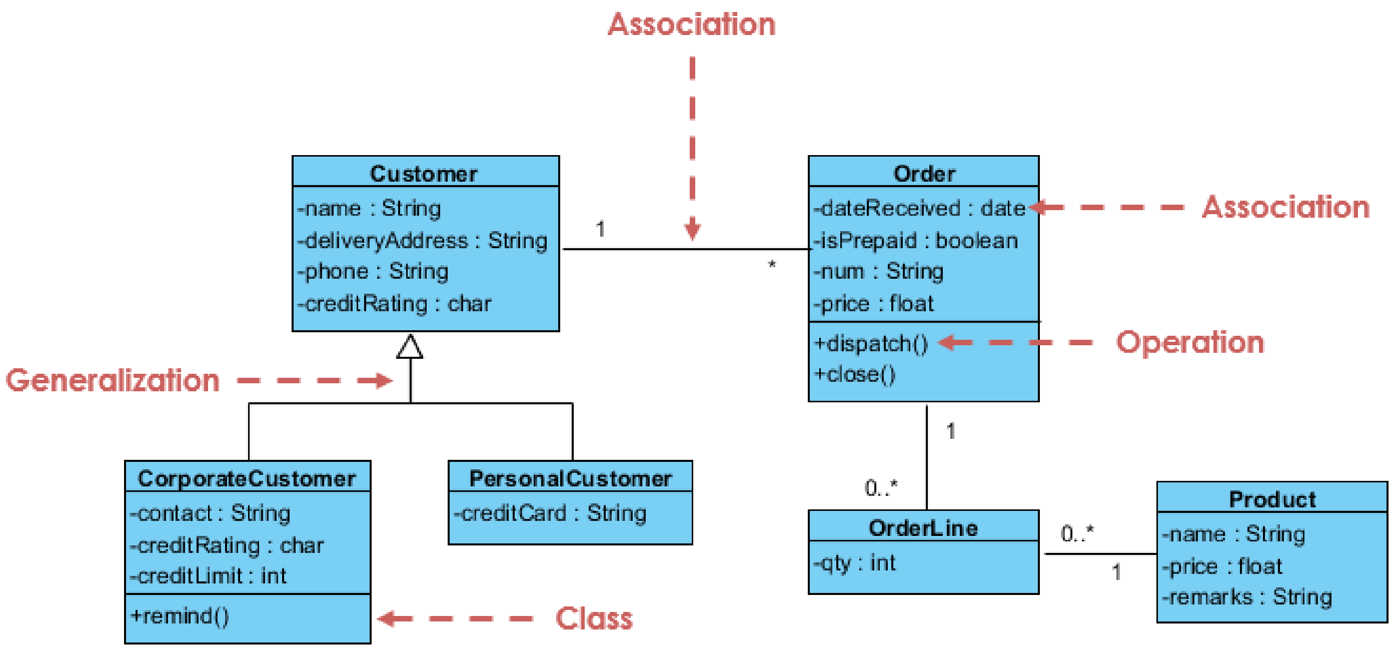
What is a Class Diagram?

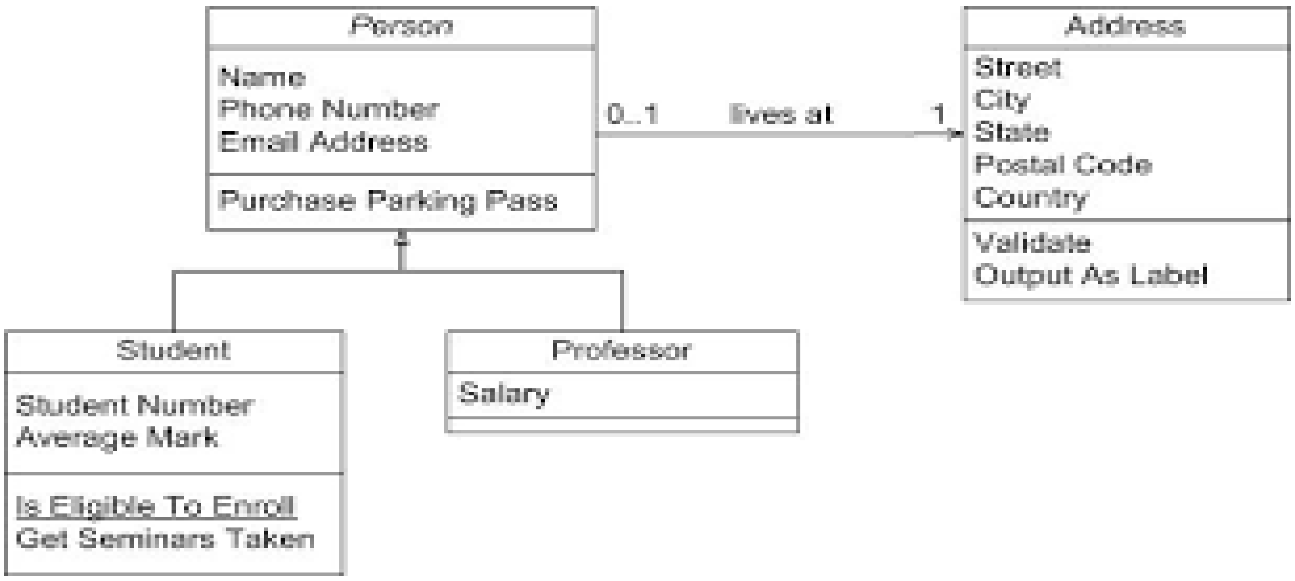
The class diagram is a central modeling technique that runs through nearly all object-oriented methods. This diagram describes the types of objects in the system and various kinds of static relationships which exist between them.
Relationships

There are three principal kinds of relationships which are important:

1. **Association** - represent relationships between instances of types (a person works for a company, a company has a number of offices.
2. **Inheritance** - the most obvious addition to ER diagrams for use in OO. It has an immediate correspondence to inheritance in OO design.
3. **Aggregation** - Aggregation, a form of object composition in object-oriented design.

Class Diagram Example

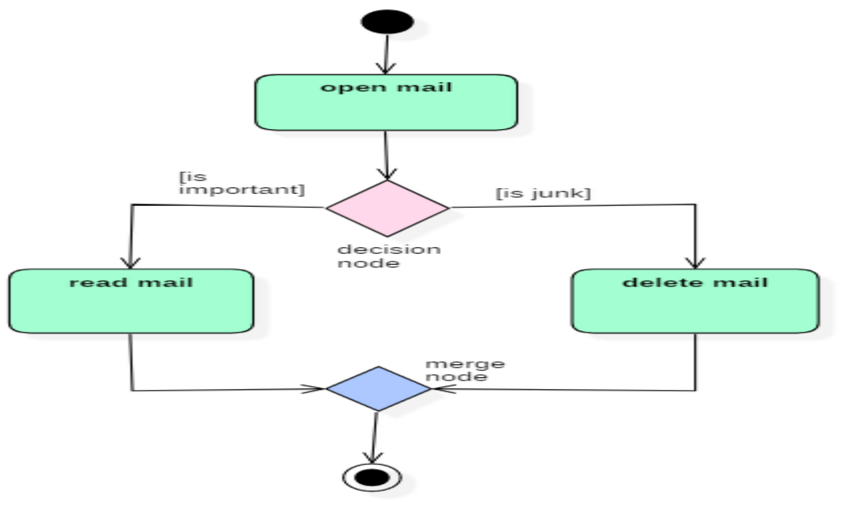




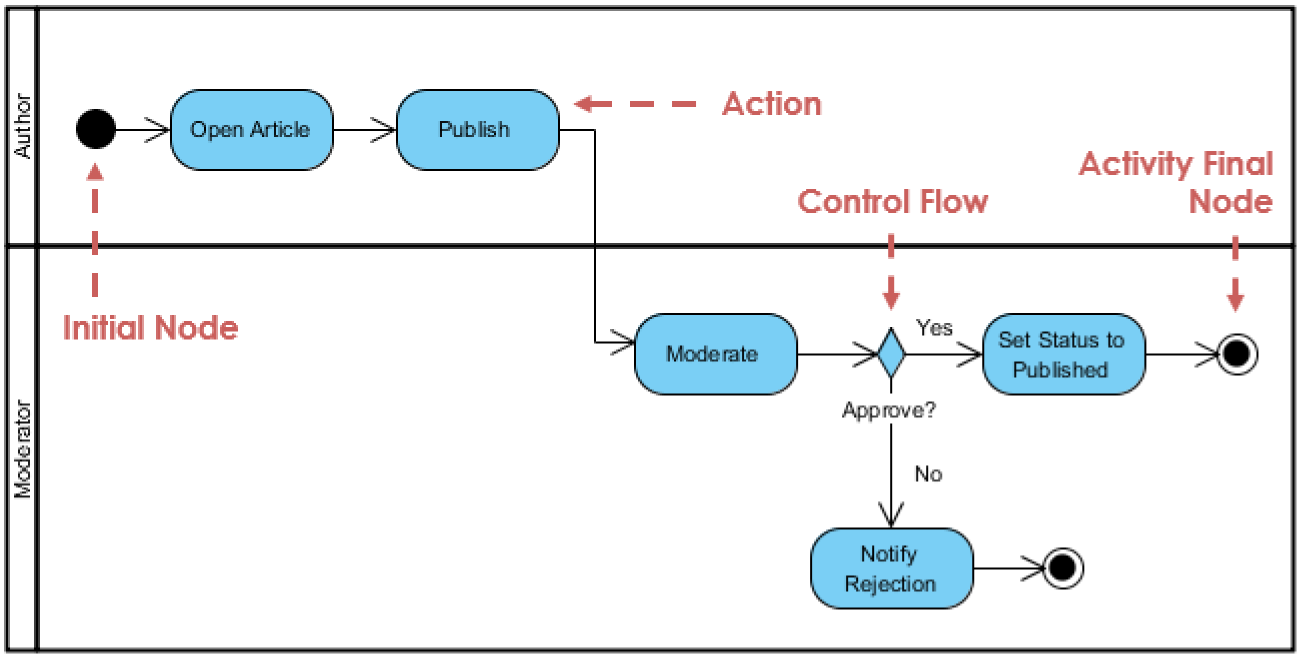
What is an Activity Diagram?

Activity diagrams are graphical representations of workflows of stepwise activities and actions with support for choice, iteration and concurrency. It describes the flow of control of the target system, such as the exploring complex business rules and operations, describing the use case also the business process.

In the Unified Modeling Language, activity diagrams are intended to model both computational and organizational processes (i.e. workflows).



Activity Diagram Example



UML tools:

There are many tools available in the market for designing UML diagrams. Example- Adobe Spark, Edraw Max, LucidChart, ConceptDraw ,Visual Paradigm,Argo UML and SmartDraw, etc.

The user interface

UML and traditional CASE tools still focus more on application internals and less on application usability aspects. A user interface (UI) is modeled in terms of its internal structure and objects comprising it, the same as the rest of the application. The adoption of use cases and interaction scenarios acknowledges the importance of recognizing user tasks when developing an application, but it is still used mainly as a starting point for designing software implementing usage scenarios rather than focusing on modeling user tasks to improve application usability. Explicit modeling of user interface domain knowledge can bring important benefits when utilized by a CASE tool: additional design assistance with exploring UI design alternatives, support for evaluating and critiquing UI designs, as well as increased reuse and easier maintenance. UML can provide a notation framework for integrating user interface modeling with mainstream software engineering OO modeling. The built-in extensibility mechanisms (stereotypes, tagged values and constraints) allow the introduction of new modeling constructs with specialized semantics for UI modeling while staying within UML.

Using UML collaborations for UI design

• Identify UI elements • Identify their responsibilities • Identify their relationships

User Interface Analysis User Interface Analysis • Create user interface realization • Identify candidate UI elements • Model role-boundary interaction • Re-factor UI responsibilities • Model UI navigation

Here we will show how to use and specialize UML diagrams in order to describe the user interface and user interactions of a software system, following a particular model driven development (MDD) perspective. Model driven development involves creating models through a methodological process that begins with requirements and looks into a high-level architectural design. Model-driven development facilitates and improves the software analysis and design and code generation facilities from models prevent the loss of substantial information during the transition of a model to its implementation.

In our MDD perspective, we consider the following steps for user interface design and modelling:

1. Firstly, we use a UML use case diagram for extracting the main user interfaces.

2. Secondly, we describe each use case by means of a special kind of UML activity diagrams, called user-interaction diagrams, whose states represent data output actions and transitions represent data input events. This perspective allows the designer to model the user interaction (i.e., input-output interaction) in each main user interface.

3. Thirdly, each input and output interaction of the user-interaction diagrams allows the designer to extract GUI components used in each user interface. Therefore, we can obtain a new and specialized version of the use case diagram representing the user interface design, and a class diagram for GUI components: user-interface and GUI-class diagrams, respectively.

4. The user-interaction, user-interface, and GUI-class diagrams can be seen as the UML-based user interface models of the system.