4. PETRI NETS

Petri nets are a mathematical modeling tool used in software engineering to analyze and describe the behavior of concurrent systems. They were introduced by Carl Adam Petri in the 1960s and have since been widely used in various fields, including software engineering.

Petri nets provide a graphical representation of a system's state and the transitions between those states. They consist of places, transitions, and arcs. Places represent states, transitions represent events or actions, and arcs represent the flow of tokens (also called markings) between places and transitions.

In software engineering, Petri nets can be used for various purposes, including:

System Modeling: Petri nets are used to model the behavior of complex software systems, especially concurrent and distributed systems. They help in understanding and visualizing how different components of the system interact and how their states change over time.

Specification and Verification: Petri nets can be used to specify the desired behavior of a software system. By modeling the system using Petri nets, it becomes possible to formally verify properties such as safety, liveness, reachability, and deadlock-freeness. Verification techniques based on Petri nets help in identifying design flaws, potential deadlocks, or other issues early inthe development process.

Performance Analysis: Petri nets can be used to analyze the performance of software systems, including their throughput, response time, and resource utilization. By modeling the system's behavior and introducing timing annotations, it becomes possible to perform quantitative analysis and predict system performance under different conditions.

Workflow Modeling: Petri nets are often used to model business processes and workflows. In software engineering, this is particularly useful for modeling the flow of tasks and activities in software development processes, such as agile methodologies or continuous integration/continuousdeployment (CI/CD) pipelines.

Software Testing: Petri nets can be utilized in software testing to generate test cases and verify the correctness of the system. By modeling the system's behavior and different scenarios, it becomes possible to systematically generate test cases that cover various paths and states, helping inidentifying potential bugs and ensuring the software's reliability.

5. OBJECT MODELLING USING UML

UML (Unified Modeling Language) can be called a **graphical modeling language** that is used in the field of software engineering. It specifies, visualizes, builds, and documents the softwaresystem's artifacts (main elements). It is a visual language, not a programming language. UML diagrams are used to depict the behavior and structure of a system. UML facilitates modeling, design, and analysis for software developers, business owners, and system architects. A pictureis worth a thousand words.

The Benefits of Using UML

- Because it is a general-purpose modeling language, it can be used by any modeler. UML is a straightforward modeling approach utilized to describe all practical systems.
- Because no standard methods were available then, UML was developed to systemize and condense object-oriented programming. Complex applications demand collaboration and preparation from numerous teams, necessitating a clear and straightforward means of communication among them.
- The **UML diagrams** are designed for customers, developers, laypeople, or anyone who wants to understand a system, whether software or non-software. Businesspeople do not understand code. As a result, UML becomes vital for communicating the system's essential requirements, features, and procedures to non-programmers. Technical details became easier to understand by non-technical people through an introduction to UML.
- When teams can visualize processes, user interactions, and the system's static structure,they save a lot of time in the long run.

Characteristics of UML

The UML has the following characteristics:

- It is a modeling language that has been generalized for various use cases.
- It is not a programming language; instead, it is a graphical modeling language that usesdiagrams that can be understood by non-programmers as well.
- It has a close connection to object-oriented analysis and design.
- It is used to visualize the system's workflow.

Conceptual Modeling

Before moving ahead with the concept of UML, we should first understand the basics of the conceptual model.

A conceptual model is composed of several interrelated concepts. It makes it easy to understand the objects and how they interact with each other. This is the first step before drawing UML diagrams.

Following are some object-oriented concepts that are needed to begin with UML:

- **Object:** An object is a real world entity. There are many objects present within a singlesystem. It is a fundamental building block of UML.

Class: A class is a software blueprint for objects, which means that it defines the variables and methods common to all the objects of a particular type.

- **Abstraction:** Abstraction is the process of portraying the essential characteristics of an object to the users while hiding the irrelevant information. Basically, it is used to envision the functioning of an object.
- **Inheritance:** Inheritance is the process of deriving a new class from the existing ones.
- **Polymorphism:** It is a mechanism of representing objects having multiple forms used for different purposes.
- **Encapsulation:** It binds the data and the object together as a single unit, enabling tight coupling between them.

Use Case Diagram

A use case diagram is used to represent the dynamic behavior of a system. It encapsulates the system's functionality by incorporating use cases, actors, and their relationships. It models the tasks, services, and functions required by a system/subsystem of an application. It depicts the high-level functionality of a system and also tells how the user handles a system.

Purpose of Use Case Diagrams

The main purpose of a use case diagram is to portray the dynamic aspect of a system. It accumulates the system's requirement, which includes both internal as well as external influences. It invokes persons, use cases, and several things that invoke the actors and elements accountable for the implementation of use case diagrams. It represents how an entity from the external environment can interact with a part of the system.

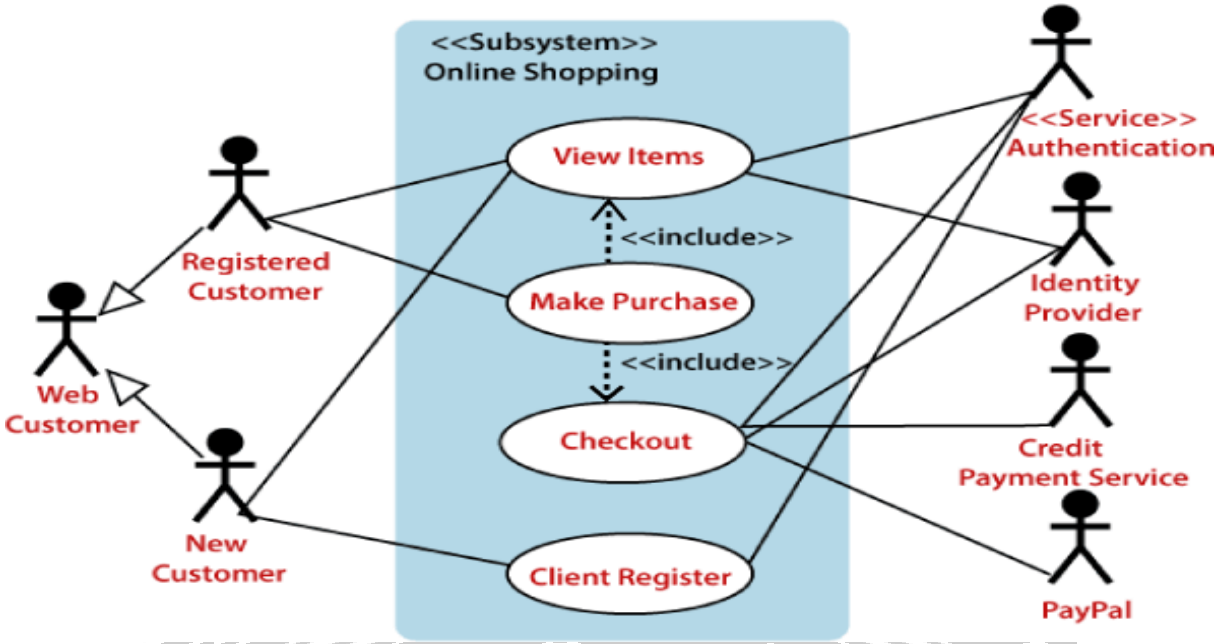
Following are the purposes of a use case diagram given below:

1. It gathers the system's needs.
2. It depicts the external view of the system.
3. It recognizes the internal as well as external factors that influence the system.
4. It represents the interaction between the actors.

Example of a Use Case Diagram

A use case diagram depicting the Online Shopping website is given below.

Here the Web Customer actor makes use of any online shopping website to purchase online. The top-level uses are as follows; View Items, Make Purchase, Checkout, Client Register. The **View Items** use case is utilized by the customer who searches and view products. The **Client Register** use case allows the customer to register itself with the website for availing gift vouchers, coupons, or getting a private sale invitation. It is to be noted that the **Checkout** is an included use case, which is part of **Making Purchase,** and it is not available by itself.



- The **View Items** is further extended by several use cases such as; Search Items, Browse Items, View Recommended Items, Add to Shopping Cart, Add to Wish list. All of these extended use cases provide some functions to customers, which allow them to search for an item.

- Both **View Recommended Item** and **Add to Wish List** include the Customer Authentication use case, as they necessitate authenticated customers, and simultaneously item can be added to the shopping cart without any user authentication.

Similarly, the **Checkout** use case also includes the following use cases, as shown below. It requires an authenticated Web Customer, which can be done by login page, user authentication

- cookie ("Remember me"), or Single Sign-On (SSO). SSO needs an external identity provider's participation, while Web site authentication service is utilized in all these use cases.

- The Checkout use case involves Payment use case that can be done either by the credit cardand external credit payment services or with PayPal.