Dr. Shipra Saraswat

UNIT-2

Basic Structural Modelling

classes

A class is a group of objects that share common properties and behaviour. For example, we can consider a class Car as a class that has characteristics like steering wheels, seats, breaks etc. And its behaviour is mobility.

Relationships

The relation between objects defines how these objects will interact or collaborate to perform an operation in an application. The relationship between objects defines how these objects will interact or collaborate to perform an operation in an application.

Common Mechanisms in UML

The UML is made simpler by the presence of four common mechanisms that apply throughout the language: specifications, adornments, Common divisions and extensibility mechanisms.

class and object Diagrams

A class and object diagram shows your classes and their relationships. An object model diagram shows the interaction between objects at some point, during run time. A class diagram will show what the objects in your system consist of number and what they are capable of doing (methods) mostly static.

Common Modelling Techniques for class and object Diagrams

→ Modeling simple Collaborations.
→ Modeling a logical database schema.
→ Forward and reverse engineering.

Modeling Techniques For object Diagrams

→ Modeling object structures.
→ Forward and reverse engineering.

Collaboration Diagrams

Collaboration Diagrams also known as Communication diagram, is an illustration of the relationships and interactions among software objects in the Unified modeling language.

Sequence Diagrams

UML Sequence Diagrams are interaction diagrams that detail how operations are carried out. They Capture the interaction between objects in the Context of a Collaboration. Sequence diagrams are time focus and they show the order of the interaction visually by using the vertical axis of the diagram to represent time what messages are sent and when.

CallBack Mechanism

Callbacks is a mechanism in OOPs that allows an application to handle subscribed events, arising at runtime, through a listener interface. The subscribers will need to provide a concrete implementation of the interface abstract methods.

this way, whenever the specified event is triggered, the predefined actions associated with it will execute in the order they were defined.

Broadcast Messages

Broadcast messages are messages that are sent by a single object but received by more than one. Broadcast messages imply concurrency. They apply to both uniprocessor systems running multiple threads of execution and multiprocessor systems. Broadcast messages originate from the source object at the same point in time, but arrive at different objects at potentially different times.

Basic Behavioral Modelling

Behavioral modelling is an approach used by companies to better understand and predict consumer actions. Behavioral modeling uses available consumer and business spending data to estimate future behavior in specific circumstances.

Use Case Diagrams

Use Case Diagrams model the behavior of a system and help to capture the requirements of the system. Use Case diagrams describe the high level functions and scope of a system. These diagrams also identify the interactions between the system and its actors.

Activity Diagrams

Activity diagram is basically a flowchart to represent the flow from one activity to another activity. The activity can be described as an operation of the system. The control flow is drawn from one operation to another. This flow can be sequential, branched or concurrent.

State Machine

State machine is a module that implements the virtual methods described above. State machine controllers are modules derived from the state machine class, which is derived from the Module class. An event is an abstract base class with one virtual Boolean method, called check, which takes no arguments.

Process and Thread

A process is a program under execution i.e an active program. A thread is a lightweight process that can be managed independently by a scheduler. Processes require more time for context switching as they are more heavy. Threads require less time for context switching as they are lighter than processes.

→ Process means any program is in execution.

→ Thread means a segment of a process.

→ Thread is a single sequence stream within a process. Threads have same properties as of the process so they are called as light weight processes.

Event and Signals

An event is the specification of a significant occurence that has a location in time and space. Anything that happens is modeled as an event in UML. Four kinds of events - signals, calls, the passing of time and a change in state. Events may be external or internal and asynchronous or synchronous.

Kinds of Events

Events may be internal or external. External events are those that pass between the system and its actors. Internal events are those which pass among the objects that live within the system. There are four kinds of events : signals, calls, the passing of time, and a change in state.

Signals

→ A signal represents an object that is dispatched asynchronously by one object and then received by another. Exceptions are an example of a kind of signal.

→ Signals may have instances, although these instances are not typically modeled explicitly. Signals may be involved in generalization relationships, enabling the modeling of hierarchies of events.

→ Signals may have attributes and operations. The attributes of a signal serve as its parameters.

Event and Signals

An event is the specification of a significant occurence that has a location in time and space. Anything that happens is modeled as an event in UML. Four kinds of events - signals, calls, the passing of time and a change in state. Events may be external or internal and asynchronous or synchronous.

Kinds of Events

Events may be internal or external. External events are those that pass between the system and its actors. Internal events are those which pass among the objects that live within the system. There are four kinds of events : Signals, Calls, the passing of time, and a change in state.

Signals

→ A signal represents an object that is dispatched asynchronously by one object and then received by another. Exceptions are an example of a kind of signal.

→ Signals may have instances, although these instances are not typically modeled explicitly. Signals may be involved in generalization relationships, enabling the modeling of hierarchies of events.

→ Signals may have attributes and operations. The attributes of a signal serve as its parameters.

Time Diagram

The timing diagram describes how an object underwent a change from one form to another. A waveform portrays the flow among the Software programs at several instances of time.

Important Key points of a Timing diagram

→ It emphasizes at that particular time when the message has been sent among objects.

→ It explains the time processing of an object in detail.

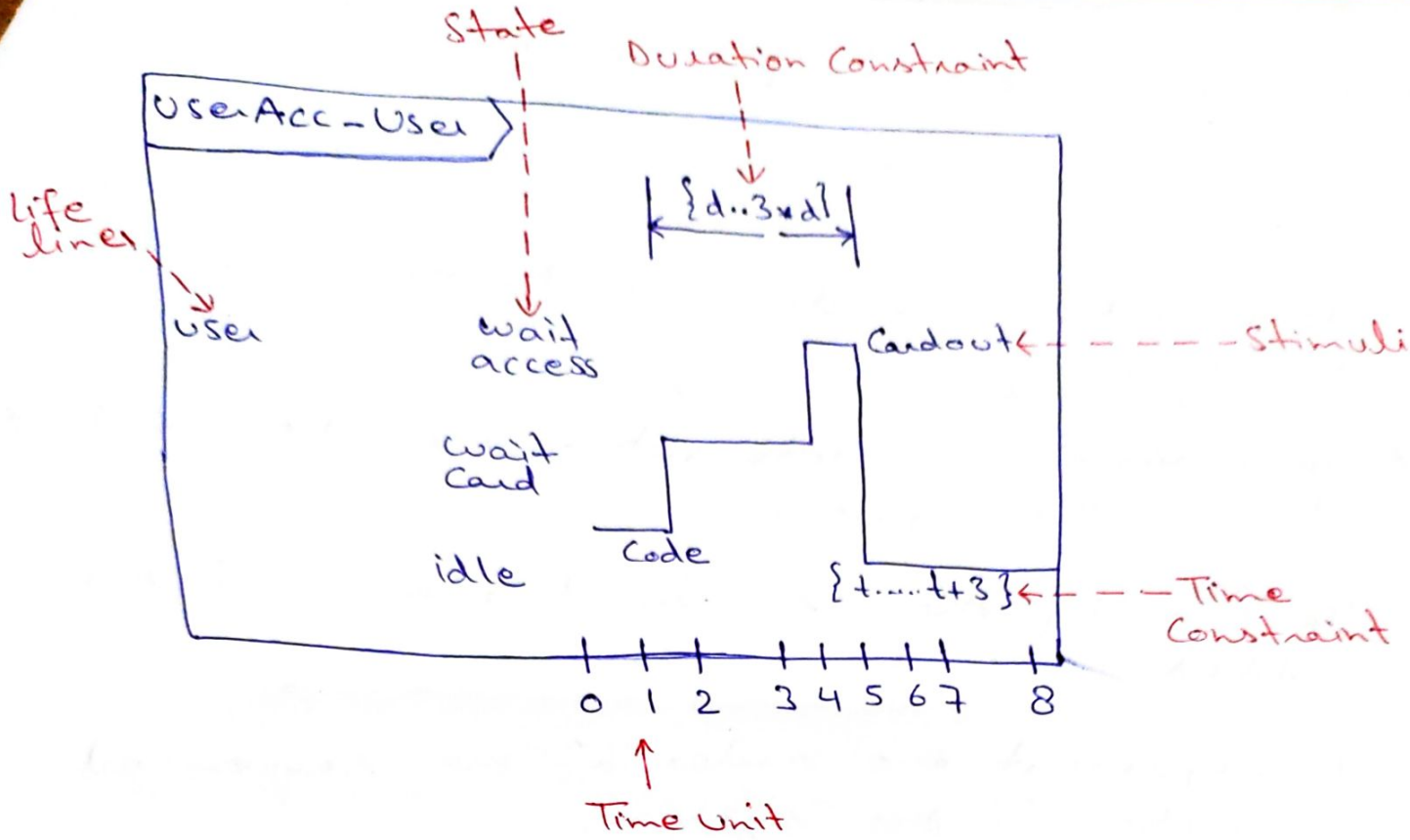
→ It is employed with distributed and embedded systems.

→ It also explains how an object undergoes changes in its form throughout its lifeline.

→ As the lifelines are named on the left side of an edge, the timing diagrams are read from left to right.

→ It depicts a graphical representation of states of a lifeline per unit time.

→ In UML, the timing diagram has come up with several notations to simplify the transition state among two lifelines per unit time.



Interaction Diagrams

Interaction diagrams are used when we want to understand the message flow and the structural organization. message flow means the sequence of control flow from one object to another. Structural organization means the visual organization of the elements in a system. To model the flow of control by time sequence.

→ The interaction diagram portrays the interactio-ns between distinct entities present in the model. It amalgamates both the activity and sequence diagrams. The Communication is nothing but units of the behavior of a classifier that provides context for interactions.

→ In interaction diagram, the critical Component is the messages and the lifeline.

Following are the purpose of an Interaction Diagram

→ To visualize the dynamic behavior of the System.

→ To envision the interaction and the message flow in the system.

→ To portray the structural aspects of the entities within the system.

→ To represent the order of the sequenced interaction in the system.

→ To visualize the real time data and represent the architecture of an object oriented system.

Package Diagram

Package diagrams are structural diagrams used to show the organization and arrangement of various model elements in the form of Packages. A package is a grouping of related UML elements, such as diagrams, documents, classes or even other packages.

Purpose of Package Diagrams

→ Package diagram can be used to simplify complex class diagrams, it can group classes into packages.

→ A package is a collection of logically related UML elements.

→ Packages are depicted as file folders and can be used on any of the UML diagrams.

Basic Concepts of Package Diagram

→ Package name should not be the same for a system, however classes inside different packages could have the same name.

→ Packages can include whole diagrams, name of components alone or no components at all.

→ Fully qualified name of a package has the following syntax.

Package Diagram - Dependency Notation

There are two sub types involved in dependency. They are << import >> & << access >>. Though there are two stereotypes users can use their own stereotype to represent the type of dependency between two packages.

Package Diagram Example – Import

<< import >> – one package imports the functionality of other package.

<< access >> – one package requires help from functions of other package.

Architectural Modeling

Architectural model represents the overall framework of the system. It contains both structural and behavioral elements of the system. Architectural model can be defined as the blueprint of the entire system. Package diagram comes under architectural modeling.

Deployment Diagrams

Deployment diagrams are used to visualize the topology of the physical components of a system, where the software components are deployed.

→ Deployment diagrams are used to describe the static deployment view of a system.

→ Deployment diagram consist of nodes and their relationships.

Purpose of Deployment Diagram

→ Visualize the hardware topology of a system.

→ Describe the hardware components used to deploy software components.

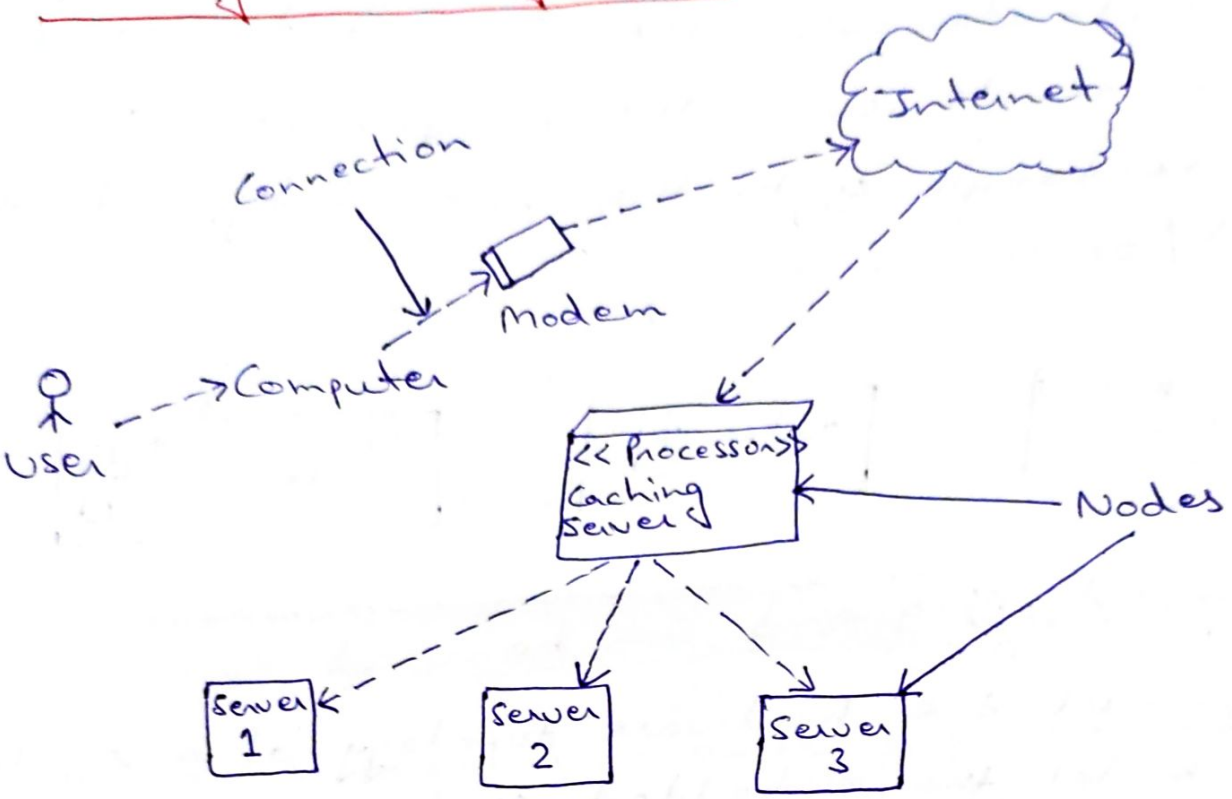
→ Describe the runtime processing nodes.

How to draw a Deployment Diagram

Deployment diagrams are useful for system engineers. An efficient deployment diagram is very important as it controls the following parameters.

1) Performance
2) Scalibility
3) Maintainability
4) Portability

Deployment Diagram of an order management system



Component Diagrams

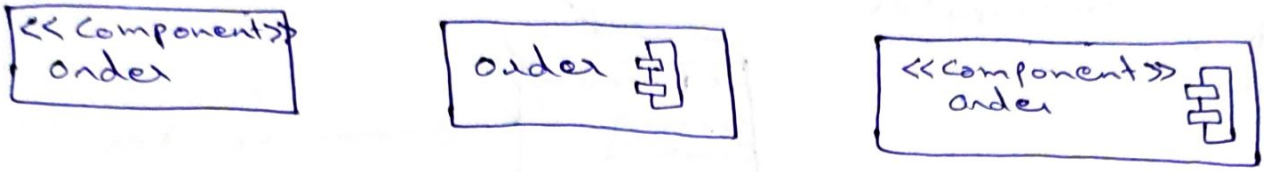
Component Diagrams also known as a UML Component diagram, describes the organization and wiring of the physical components in a system. Component diagrams are often drawn to help model implementation detail and double check that every aspect of the system's required functions is covered by planned development.

Basic Concepts of Component Diagram

A component represents a modular part of a system that encapsulates its contents and whose manifestation is replaceable within its environment. In UML, a component is drawn as a rectangle with optional compartments stacked vertically. A high level, abstracted

View of a Component in UML 2 can be modeled as :

→ A rectangle with the Components name
→ A rectangle with the Component icon
→ A rectangle with the stereotype text and / or icon.



Deployment Diagrams can be used

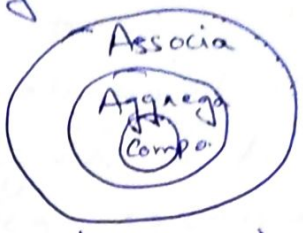
→ To model the hardware topology of a system.
→ To model the embedded system.
→ To model the hardware details for a client / Sever system.
→ To model the hardware details of a distributed application.
→ For Forward and reverse engineering.

Component Diagrams

Component diagrams describes the organization and wiring of the physical Components in a system. Component diagrams are often drawn to help model implementation details and double check that every aspect of the system's required functions is covered by planned development.

Relationships

Graphically, a component diagram is a collection of vertices and arcs and commonly contain components, interfaces and dependency, aggregation constraint, generalization, association, and realization relationships. It may also contain notes and constraints.



Association (Has a relationship)

→ An association specifies a semantic relation ship that can occur between typed instances.

→ It has at least two ends represented by properties, each of which is connected to the type of the end. More than one end of the association may have the same type.

Composition

The life span of an individual object is depends on the life span of aggregate object

→ Composite aggregation is a strong form of aggregation that requires a part instance be included in at most one composite at a time.

→ If a Composite is deleted, all of its parts are normally deleted with it.

→ is-a relationship.

Aggregation (Has a relationship between classes)

A kind of association that has one of its end marked shared as kind of aggregation, meaning that it has a shared aggregation.

→ It is a specialized form of association between two or more objects in which each object has its own life cycle but there exists an ownership as well.

Constraint

A condition or restriction expressed in natural language text or in a machine readable language for the purpose of declaring some of the semantics of an element.

Dependency

→ A dependency is a relationship that signifies that a single or a set of model elements requires other model elements for their specification or implementation.

→ This means that the complete semantics of the depending elements is either semantically or structurally dependent on the definition of the supplier.

Links

→ A generalization is a taxonomic relationship between a more general classifier and a more specific classifier.

→ Each instance of the specific classifier is also an indirect instance of the general classifier.

→ Thus, the specific classifier inherits the features of the more general classifier.

Static Member function of a class

A static member function can be called even if no objects of the class exist and the static functions are accessed using only the class name and the scope resolution operator. A static member function can only access static data member, other static member functions and any other functions from outside the class.

Static members of a class

Static data members are class members that are declared using static keywords. A static member has certain special characteristics. These are : only one copy of that member is created for the entire class and is shared by all the objects of that class, no matter how many objects are created.

Static Function in OOPs

Static functions are used to invoke a class code when none of its instance exists. Static functions can change static variables.

characteristics of Static Data member :-

1) It is initialized to zero when the first object of its class is created.
2) It is visible only within the class, but its lifetime is the entire program.
3) only one copy of that member is created for the entire class and it is shared by all the objects of that class

Characteristics of Static data member in a class

Static data members are class members that are declared using static keywords. A static member has certain special characteristics. These are : only one copy of that member is created for the entire class and is shared by all the objects of that class, no matter how many objects are created.

Static data member can accessed through the member functions, but the function should be Static member function.

A static member function is a special member function, which is used to access only static data members, any other normal data member cannot be accessed through static member function. Just like Static data member, Static member function is also a class function; it is not associated with any class object. we can access a static member function with class name, by using following Syntax:-

```
class_name :: function_name (perameter);
```

Consider the example

```
#include<iostream>
using namespace std;
class Demo
{
  private:
      // Static data members
      Static int X;
      Static int Y;
```

```
    Public :
    // Static member function
    Static void Print ()
    {
        Cout << "Value of x : " << x << endl;
        Cout << "Value of y : " << y << endl;
    }
};
// Static data members initializations
int Demo :: x = 10;
int Demo :: y = 20;

int main ()
{
    Demo OB;
    // accessing class name with object name
    Cout << " Printing through object name : " << endl;
    OB. Print ();

    //accessing class name with class name
    Cout << " Printing through class name : " << endl;
    Demo :: Print ();

    return 0;
}
```

Output

```
Printing through object name :
Value of x : 10
Value of y : 20

Printing through class name :
Value of x : 10
Value of y : 20
```

In above program X and Y are two static data members and print() is a static member function. According to the rule of static in C++, only static member function can access static data members. Non static data member can never be accessed through static member functions.

Note :- Inline function can never be static.

Why OOPs is important for Software Industries

OOPs language allows to break the program into the bit sized problems that can be solved easily (one object at a time). The new technology promises greater programmer productivity, better quality of software and lesser maintenance cost. OOPs systems can be easily upgraded from small to large systems.

→ It helps to write cleaner code and to maintain control over the functions and modules. This approach gives importance to functions rather than data. It focuses on the development of large software applications, for example, C was used for modern operating system development.

Importance of OOPs in Real Life

Following are the simple explanation of object oriented concepts with a real life example.