OBJECT ORIENTED SYSTEM DESIGN

(KCS - 054)

ABES Engineering College, Ghaziabad

B. Tech Odd Semester

UNIT - 1

TOPIC:

Introduction: The meaning of Object Orientation, object identity, Encapsulation, information hiding, polymorphism, generosity, importance of modelling, principles of modelling, object oriented, modelling, Introduction to UML, conceptual model of the UML, Architecture.

Object Oriented System Design

Introduction

Systems development is the process of defining, designing, testing, and implementing a new software application or program. It helps to create an efficient system and identify the areas of improvement.

Programming is used as a tool for System Development.

Studying OOSD is essential because it offers a structured and efficient approach to software development. Understanding the principles of Object Orientation and Object-Oriented Modeling provides a solid foundation for creating modular, maintainable, and scalable software systems. It promotes better code organization and fosters a natural mapping between real-world entities and software entities.

Need of Object-Oriented Design

The main aim of Object-Oriented Design (OOD) is to improve the quality and productivity of system analysis and design by making it more usable. It identifies the objects in problem domain, classifying them in terms of data and behavior.

Studying OOSD is essential because it offers a structured and efficient approach to software development. Understanding the principles of Object Orientation and Object-Oriented Modeling provides a solid foundation for creating modular, maintainable, and scalable software systems. It promotes better code organization and fosters a natural mapping between real-world entities and software entities.

Detail Description of Object-Oriented System Design

The goal of System Design is to create a design that meets the user requirements and supports the business processes. There are mainly two approaches of system / software development, *procedural and object-oriented approach.*

Procedure Oriented Approach

In procedure-based programming language, a main program can be divided into manyfunctions where each function is connected to another function to accomplish the overall goalof the program. To perform any particular task, set of function are compulsory.

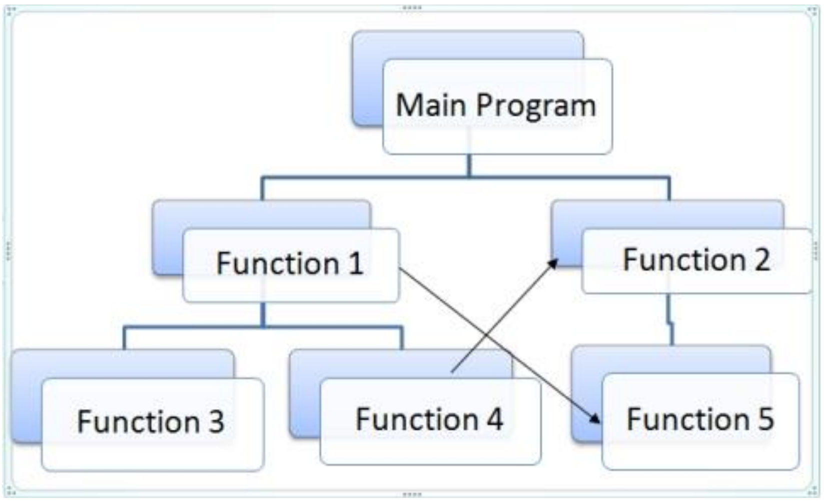


Fig 1. Procedure-Oriented Programming approach

For example, a program may involve collecting data from user, performing some kind of calculation on that data and printing the data on screen when is requested. Calculating, reading or printing can be written in a program with the help of different functions on different tasks.

Data Security Problem in Procedural Approach:

In the procedural approach some data is local i.e., within the scope of the function, while there is a need to keep some data as global which is shared among all the modules. So, thereare chances of accidental modification of global data that may violate the security of data.

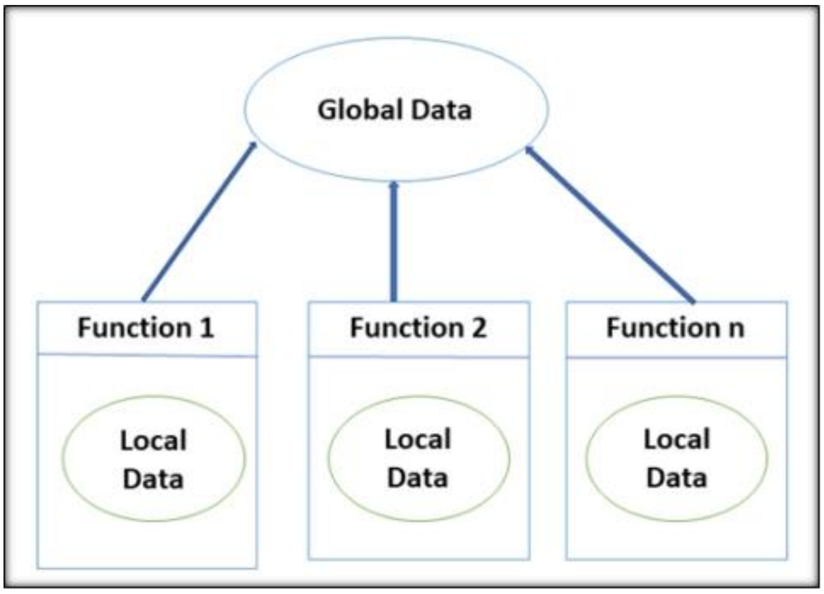


Fig 2. Local vs. Global data

Each function has its own local data and common global data, as shown in Fig. 2.

Limitations of Procedural Approach:

- The program code is harder to write when Procedural Programming is employed.
- The Procedural code is often not reusable, which may pose the need to recreate thecode if is needed to use in another application
- Difficult to relate with real-world objects
- The importance is given to the operation rather than the data, which might pose issuesin some data-sensitive cases
- The data is exposed to the whole program, making it not so much security friendly

Object Oriented Approach (OOP)

An OOP method differs from POP in its basic approach itself. All the best features of structured of OOP is developed by retaining the programming method, in which they have added number of concepts which makes efficient programming. Object oriented programming methods have number of features and it makes possible an entirely new way of approaching a program. This has to be noted that OOP retains all best features of POP method like functions/sub routines, structure etc.

The Advantages And Disadvantages Of OOP are:-

Advantages of OOP:

1. **Reusability:** OOP allows developers to create code that can be reused in different parts of an application. This makes development faster and more efficient because developers do not have to write new code from scratch each time they need to create a new feature.
2. **Modularity:** OOP allows developers to break down complex systems into smaller, more manageable modules. This makes it easier to develop, test, and maintain code because changes made to one module do not affect other parts of the system.
3. **Encapsulation:** OOP allows developers to hide the implementation details of objects, making it easier to change the behavior of an object without affecting other parts of the system. This also improves security by limiting access to sensitive data.
4. **Inheritance:** OOP allows developers to create new classes by inheriting characteristics from existing classes. This reduces the amount of code that needs to be written and makes it easier to maintain the codebase.

Disadvantages of OOP:

1. **Steep Learning Curve:** OOP is a complex paradigm, and it can take time for developers to become proficient in it. The concepts of inheritance, polymorphism, and encapsulation can be difficult to understand for beginners.

2. **Overhead:** OOP code can be more verbose than code written in other paradigms, which can result in slower performance. Additionally, OOP often requires more memory and processing power than other paradigms.

3. **Complexity:** OOP can lead to complex code, especially when dealing with large systems that have many interdependent objects. This complexity can make it more difficult to debug and maintain code.

4. **Limited Reusability:** Although OOP allows for code reusability, it can also lead to tightly coupled code that is difficult to reuse in other contexts. This can make it challenging to maintain the codebase in the long run.

Difference between Object Oriented Approach and Procedure Oriented Approach

S. N.	Features	Object Oriented Approach (OOP)	Procedure Oriented Approach (POP)
1	**Definition**	OOP stands for Object Oriented Programming.	POP stands for Procedural Oriented Programming.
2	**Approach**	OOP follows bottom-up approach – focused on building blocks and components.	POP follows top-down approach - focused on inputs and outputs
3	**Division**	A program is divided to objects and their interactions.	A program is divided into functions and they interact.
4	**Inheritance Supported**	YES	NO support
5	**Access control**	Access control is supported via access modifiers.	No access modifiers aresupported.
6	**Data Hiding**	Encapsulation is used to hide data.	No data hiding present. Data is globally accessible.
7	**Example**	C++, Java, Python, Ruby, GoLang	C, Pascal

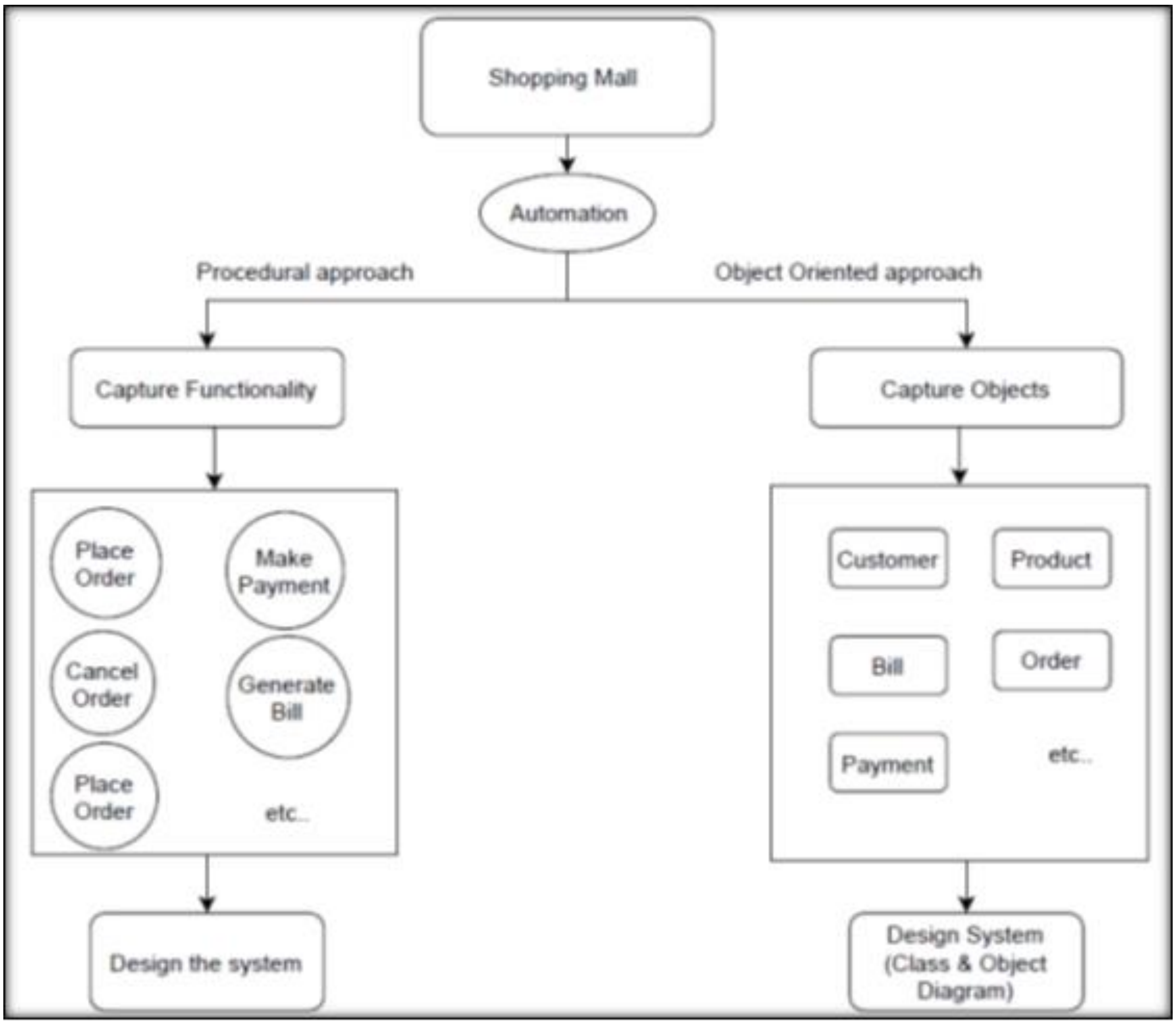


Fig 3. Procedure Oriented Vs. Object Oriented

Top-Down vs Bottom-Up Approach

In the top-down model, the system is described in general terms without addressing any specifics. Each component was then further developed, defined, and polished until the overall specification was sufficiently detailed to validate the model.

The problem is divided into parts and then parts are divided into parts so that each part can be easily solved. Thus, it uses the decomposition approach. This approach is generally used by structured programming languages such as C, COBOL, FORTRAN. The disadvantage of the top-down technique is that since each component of the code is produced independently, there may be redundancy.

The object-oriented programming approach treats the real-world problem in terms of objects. These objects work together to reach the solution of a particular problem. This typeof object-oriented programming follows the bottom-up approach. In bottom-up approach component parts are implemented first then overall system is implemented by integrating the component parts.

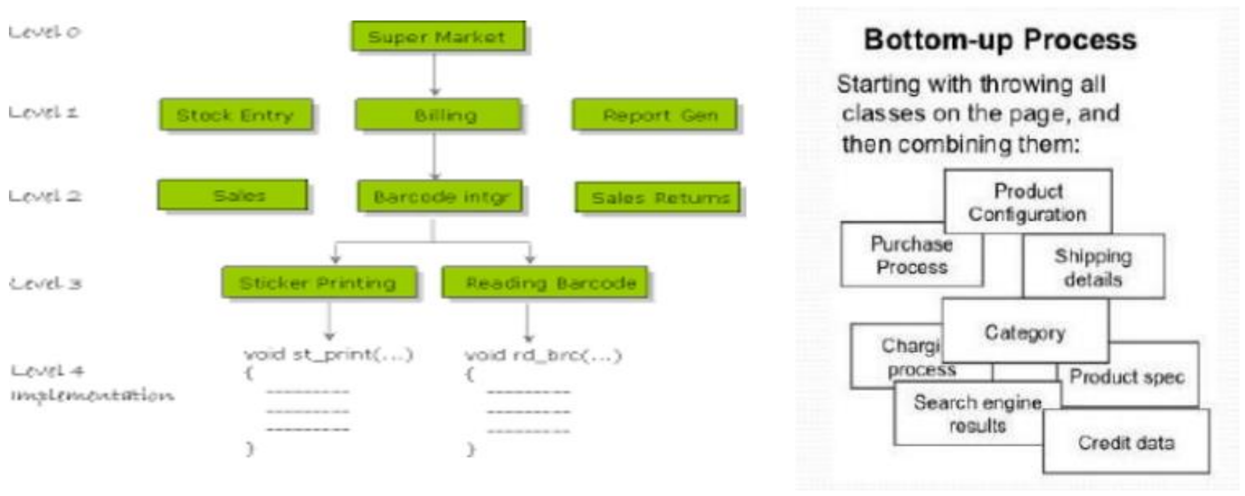
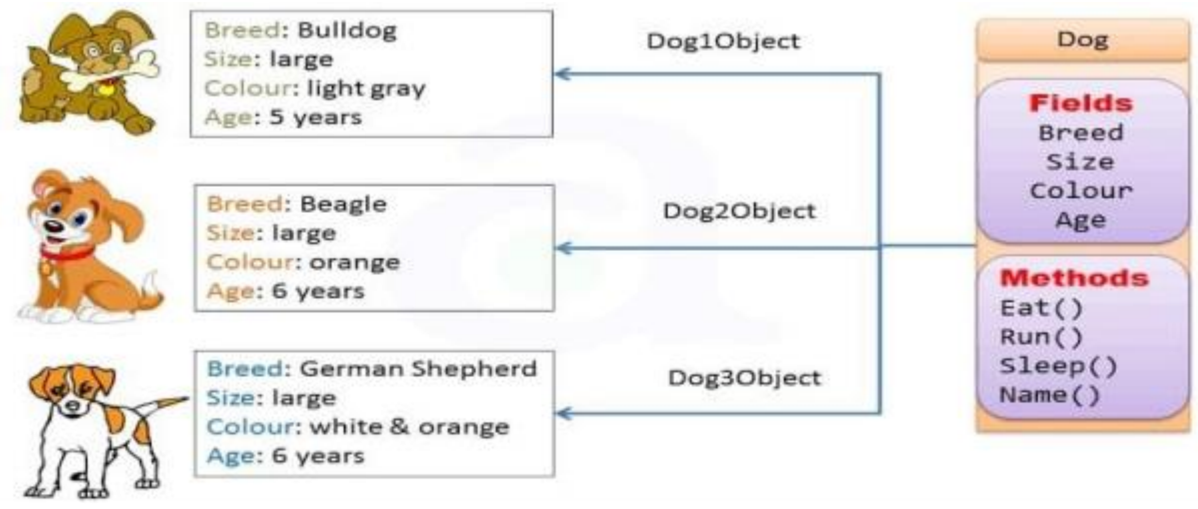


Fig. 4. Top-Down approach vs Bottom-up Approach

The Meaning of Object Orientation:

The object-oriented approach, however, focuses on objects that represent abstract or concrete things in the real world. These objects are first defined by their character and their properties, which are represented by their internal structure and their attributes (data). The behaviour of these objects is described by methods (functions).



Objects form a capsule, which combines the characteristics with behaviour. Objects are intended to enable programmers to map a real problem and its proposed software solution on a one-to-one basis.

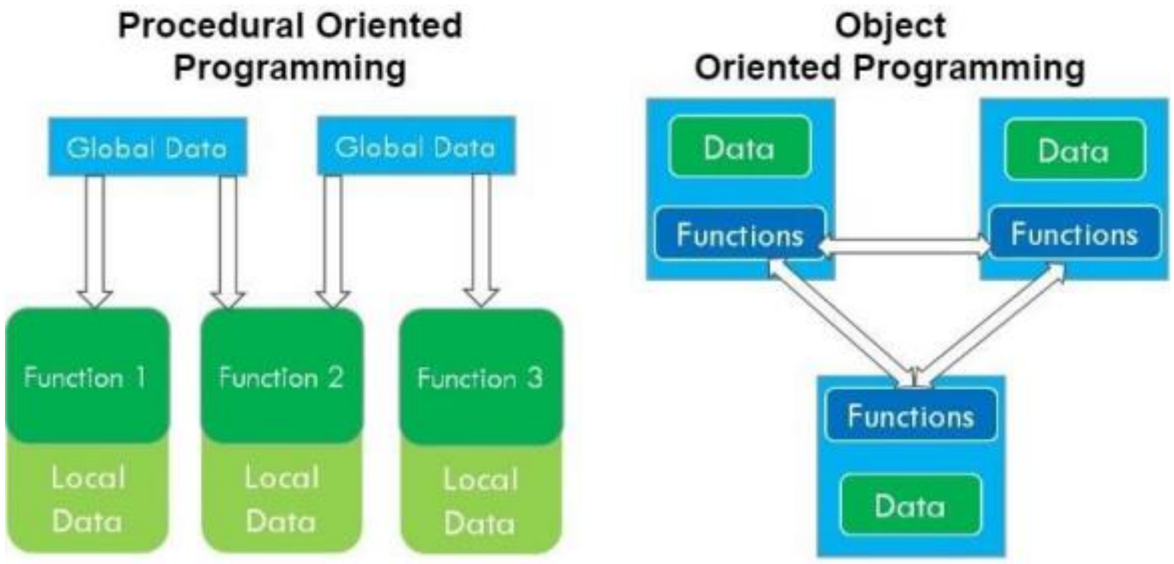


Fig 4. POP and OOP

Object examples can be, 'Customer', 'Order', or 'Invoice'.

Concepts of OOP

The OOPs concepts helps the programmers to analyse the real-world problem and implementit in terms of programming language.

Features of Object Oriented Programming

Object

An object is anything in the real-world. It exists in the form of physical or abstract. For example, it may represent a person, animal, bird or anything. For example: Elephant, Lion is an object.

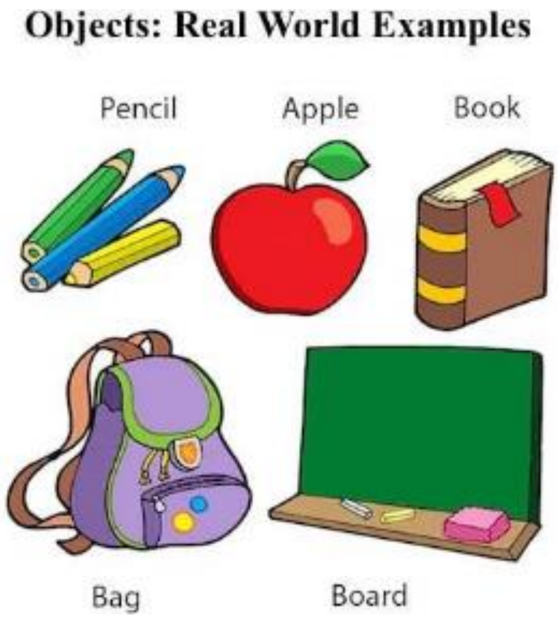


Fig. 5. Objects (Real World Example)

Object Identity

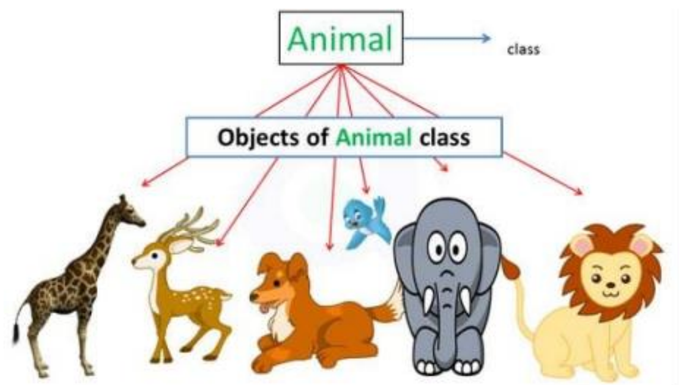
Object identity is a fundamental object orientation concept. Object Identity describes the property of Objects that distinguishes it from other Objects. Identity is a property of an object that distinguishes the object from all other objects in the application.

Object	Identity	Identity
	Car Name: **Audi** Engine Number: **a123**	Car Name: **Audi** Engine Number: **a125**

Fig. 6. Object Identity

Class

Class is a group of similar types of objects, having similar features and behaviour. In the following Figure different objects exists in the real world and based on the common features these are categorized into different classes Car, Flower and Person.



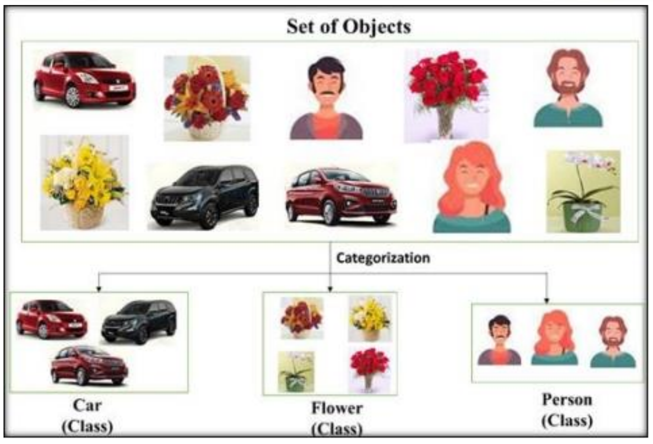


Fig. 7 Class and Object

Class is used to describe the structure of objects that how the objects will look likes. Class isalso a pattern or template which produces similar kind of objects. Class has data members (attributes) and behavior shown by object also called functionality.

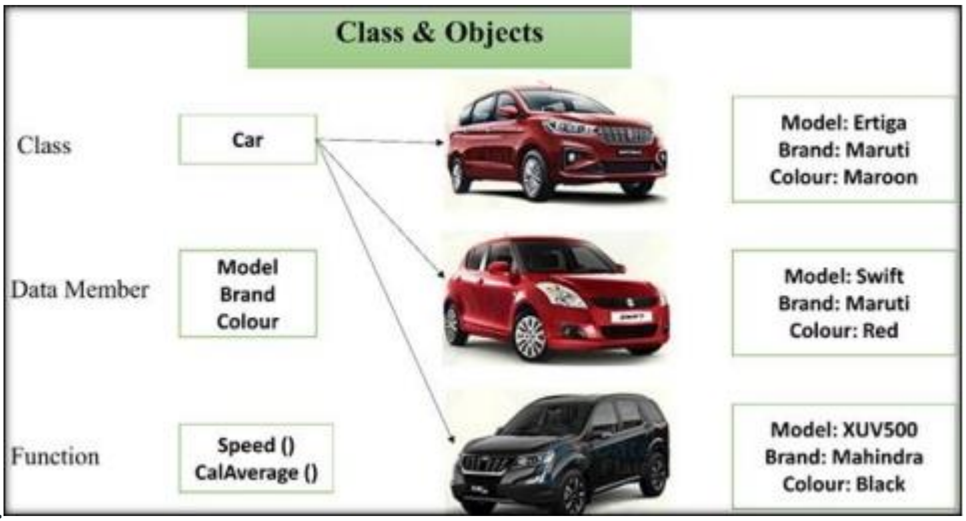


Fig.8 Data-Member and Functions of a Class

Encapsulation

Encapsulation is defined as the wrapping up of data under a single unit. It is the mechanism that binds together code and the data it manipulates. Another way to think about encapsulation is, it is a protective shield that prevents the data from being accessed by the code outside this shield.

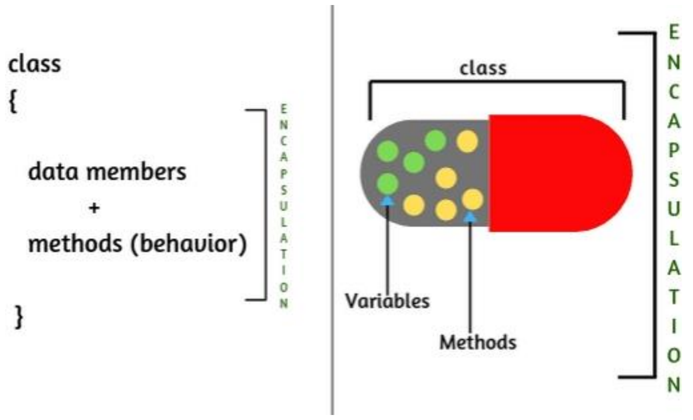


Fig.9. Encapsulation

Examples of Encapsulation:

- School bag is one of the most real examples of Encapsulation. School bag can keep ourbooks, pens, etc.
- In network communication, the data is encapsulated in a single unit in the form of packets. These packets are delivered over the network.

class Book

String book_title;
String author_name;

+ openBook()
+ closeBook()
+ readBook()

Fig. 10. Encapsulation in Object-Oriented Approach:

In this data and methods are encapsulated in a class Book.

Advantages of Encapsulation:

- The encapsulated code is more flexible and easier to change with new requirements.
- It prevents the other classes to access the private fields.
- Encapsulation allows modifying implemented code without breaking other code thathas implemented the code.
- It keeps the data and codes safe from external inheritance. Thus, Encapsulation helpsto achieve security.
- It improves the maintainability of the application.

Information Hiding

Information hiding is a powerful OOP feature. Information hiding is closely associated with encapsulation. Information or data hiding is a programming concept which protects the data from directmodification by other parts of the program.

Information hiding includes a process of combining the data and functions into a single unitto conceal data within a class by restricting direct access to the data from outside the class. "Hiding object details (state + behaviour) of a class from another class"

Real Life Example of Information Hiding:

1. My Name and personal information is stored in My Brain, nobody can access this information directly. For getting this information you need to ask me about it and it will be up to myself that how much details I would like to share with you.

2. Facebook may have millions of user accounts. Facebook is just like a Class, it has Private, Public and Protected members. In private, there may be inbox, some personalphoto or

3. video. In public, it may be some post, or user information like d.o.b, where you live etc and in protected there may be some post that only your friend can see, and it's hidden from public.

In object-oriented approach we have objects with their attributes and behaviours that are hidden from other classes, so we can say that object-oriented programming follows the principle of information hiding.

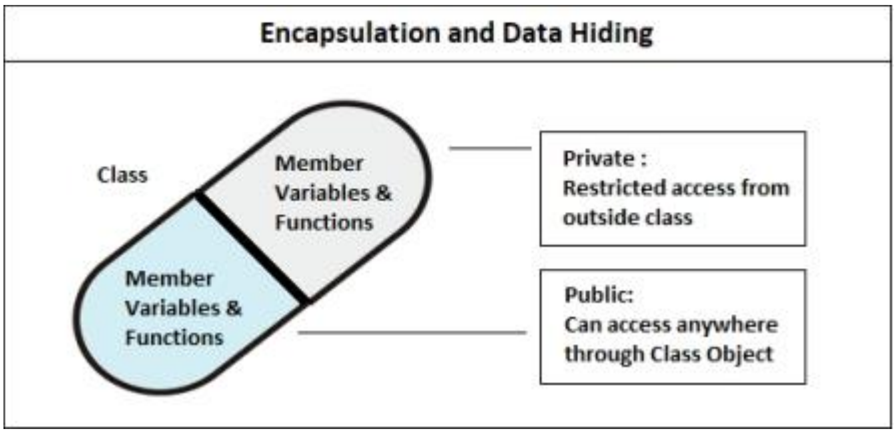

Encapsulation and Data Hiding
Class
Member Variables & Functions
Member Variables & Functions
Private : Restricted access from outside class
Public: Can access anywhere through Class Object

Fig: 11 Encapsulation & Data Hiding

Demo of information hiding in C++:

```
class  Student
{
private:
          char   name[30];
          int     marks;
 public:
};
void   display();
int     main()
{
     Student   s;
     s.marks  =  50;   //  Compilation  Error  –  Not  Accessible  as  the  member  is  private
     return     0;
}
```

The accessibility often plays an important role in information hiding. Here the data element *marks* are private element and thus it cannot be accessed by main function or any other function except the member function *display()* of class student. To make it accessible in mainfunction, it should be made a *public* member.

Advantages of Information Hiding:

1. It ensures exclusive data access and prevents intended or unintended changes in thedata.
2. It helps in reducing system complexity and increase the robustness of the program.
3. It heightens the security against hackers that are unable to access confidential data.
4. It prevents programmers from accidental linkage to incorrect data.

Disadvantage of Information Hiding:

It may sometimes force the programmer to use extra coding for creating effects for hiding thedata.

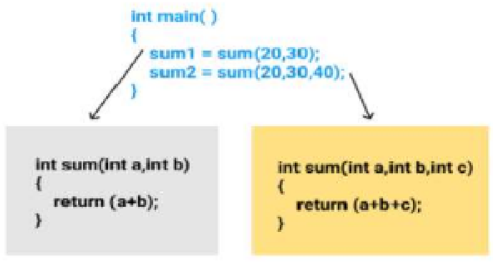
Abstraction

- Abstraction means displaying only essential information and hiding the implementation details or background details.

Polymorphism

- The word polymorphism means having many forms.
- In simple words, we can define polymorphism as the ability of a message to be displayed in more than one form.

Example of polymorphism



```
int main( )
{
   sum1 = sum(20,30);
   sum2 = sum(20,30,40);
}
```

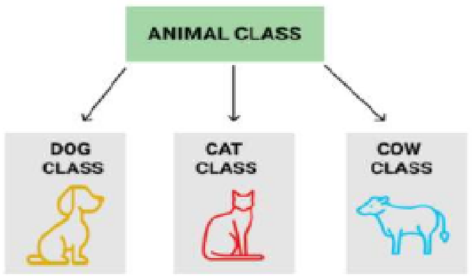
```
int sum(int a,int b)
{
   return (a+b);
}
```

```
int sum(int a,int b,int c)
{
   return (a+b+c);
}
```

<u>Inheritance</u>

- The capability of a class to derive properties and characteristics from another class is called Inheritance.
- **Sub Class**: The class that inherits properties from another class is called Sub class or Derived Class.
- **Super Class**:The class whose properties are inherited by sub class is called Base Class or Super class.
- **Reusability**: Inheritance supports the concept of "reusability", i.e. when we want to create a new class and there is already a class that includes some of the code that we want, we can derive our new class from the existing class. By doing this, we are reusing the fields and methods of the existing class.

Example: Dog, Cat, Cow can be Derived Class of Animal Base Class.



Inheritance is the mechanism that permits new classes to be created out of existing classes by extending and refining its capabilities. The existing classes are called the base classes/parent classes/super-classes, and the new classes are called the derived classes/child classes/subclasses.

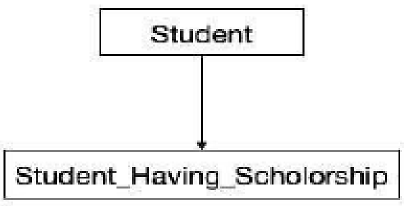
The subclass can inherit or derive the attributes and methods of the super-class(es) provided that the super-class allows so. Inheritance defines an "is – a" relationship.

Types of Inheritance:

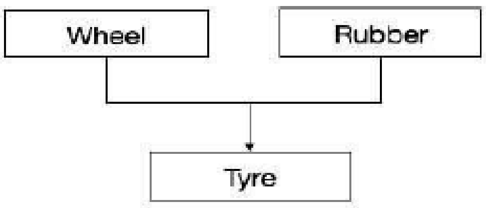
- **Single Inheritance** : A subclass derives from a single super-class.

- **Multiple Inheritance** : A subclass derives from more than one super-classes.
- **Multilevel Inheritance** : A subclass derives from a super-class which in turn is derived from another class and so on.
- **Hierarchical Inheritance** : A class has a number of subclasses each of which may have subsequent subclasses, continuing for a number of levels, so as to form a tree structure.
- **Hybrid Inheritance** : A combination of multiple and multilevel inheritance so as to form a lattice structure.

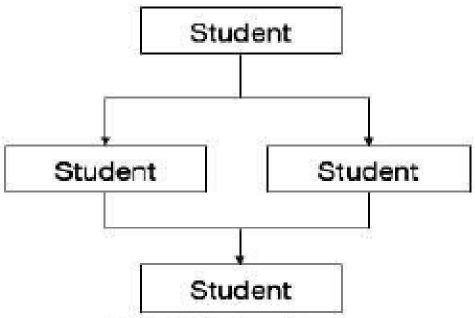
The following figure depicts the examples of different types of inheritance.



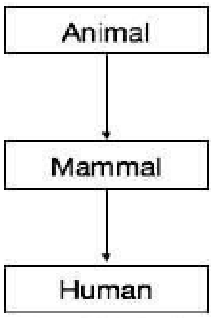
Single Inheritance



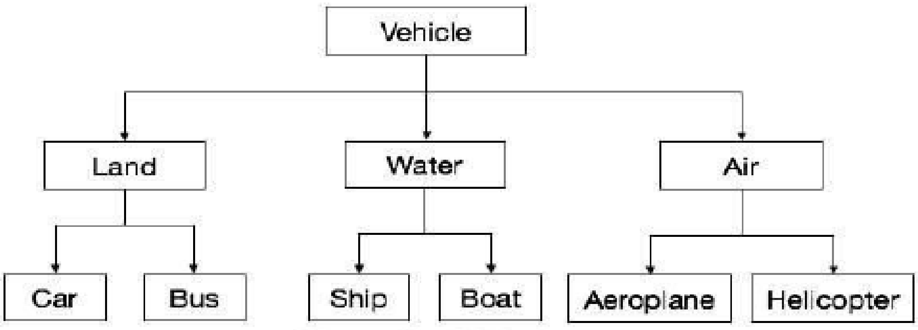
Multiple Inheritance



Hybrid Inheritance



Multi-level Inheritance



Hierarchical Inheritance

<u>**Dynamic binding**</u>

- In dynamic binding, the code to be executed in response to function call is decided at runtime
- this is also called dynamic binding or late binding or run-time binding.
- Dynamic binding is achieved using virtual functions.
- Base class pointer points to derived class object. And a function is declared virtual in base class, then the matching function is identified at run-time using virtual table entry.

<u>**Message passing**</u>

- Objects communicate with one another by sending and receiving information to each

other.
- A message for an object is a request for execution of a procedure and therefore will invoke a function in the receiving object that generates the desired results.
- Message passing involves specifying the name of the object, the name of the function and the information to be sent.

Conventional Programming
- Conventional programming writing a program in a traditional procedural language such as assembly language or high level compiler language (C , Pascal , COBOL , FORTRAN ,etc).
- The **disadvantage** is that it doesn't help to manage program complexity at scale.
- As your application grows substantially in size, it becomes more and more difficult to maintain the structure of your program in an understandable fashion, in a way that makes it easier to extend functionality.

Advantage of OOP
- Data Re-usability
- Data Redundancy
- Easy Maintenance
- Data hiding
- Security

Object Oriented Language.
- The first object-oriented programming language, the most popular OOP languages are:
- Java
- JavaScript
- Python
- C++
- Visual Basic .NET
- Ruby
- Scalable(scala)
- PHP

Introduction to 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 modelling, design and analysis. The Object Management Group (OMG) adopted Unified Modelling 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.

Three Aspects of UML:

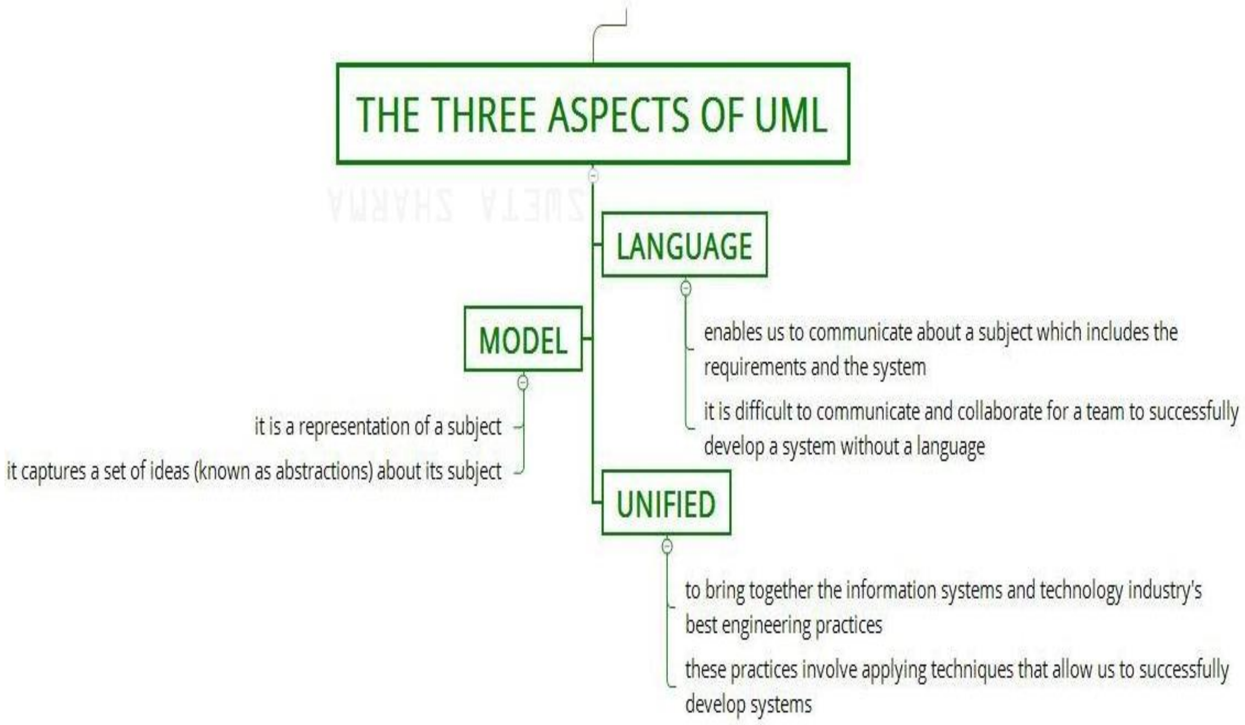

THE THREE ASPECTS OF UML
LANGUAGE
enables us to communicate about a subject which includes the requirements and the system
it is difficult to communicate and collaborate for a team to successfully develop a system without a language
MODEL
it is a representation of a subject
it captures a set of ideas (known as abstractions) about its subject
UNIFIED
to bring together the information systems and technology industry's best engineering practices
these practices involve applying techniques that allow us to successfully develop systems

Figure – Three Aspects of UML

1. Language:

It enables us to communicate about a subject which includes the requirements and the system.

It is difficult to communicate and collaborate for a team to successfully develop asystem without a language.

2. Model:

It is a representation of a subject.

It captures a set of ideas (known as abstractions) about its subject.

3. Unified:

It is to bring together the information systems and technology industry's best engineering practices.

These practices involve applying techniques that allow us to successfully develop systems.

Goals of UML:

There are a number of goals for developing UML but the most important is to define some general-purpose modelling language, which all modellers can use and it also needs to be made simple to understand and use.

UML diagrams are not only made for developers but also for business users,
common people, and anybody interested to understand the system.
The system can be a software or non-software system. Thus it must be clear that UML is not a development method rather it accompanies with processes to make it a successful system.

Where Can the UML Be Used?

The UML is intended primarily for software-intensive systems. It has been used effectively
for such domains as

• Enterprise information systems

• Banking and financial services• Telecommunications

• Transportation

• Defence/aerospace

• Retail

• Medical electronics

• Scientific

• Distributed Web-based service

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 – Abstraction in UML refers to the process of emphasizing the essential aspects of a system or object while disregarding irrelevant details. By abstracting away unnecessary complexities, abstraction facilitates a clearer understanding and communication among stakeholders.

5.Encapsulation – Binding data 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.

Conceptual model of the UML:

A conceptual model needs to be formed by an individual to understand UML. UML contains three types of building blocks**:** things, relationships, and diagrams.

1. Things

– Structural things: Classes, interfaces, collaborations, use cases, components, and nodes.

– Behavioural things: Messages and states.

– Grouping things: Packages

– Annotation things: Notes

2. Relationships: Dependency, Association, Generalization and Realization.

3. Diagrams: class, object, use case, sequence, collaboration, state chart, activity,component and deployment.

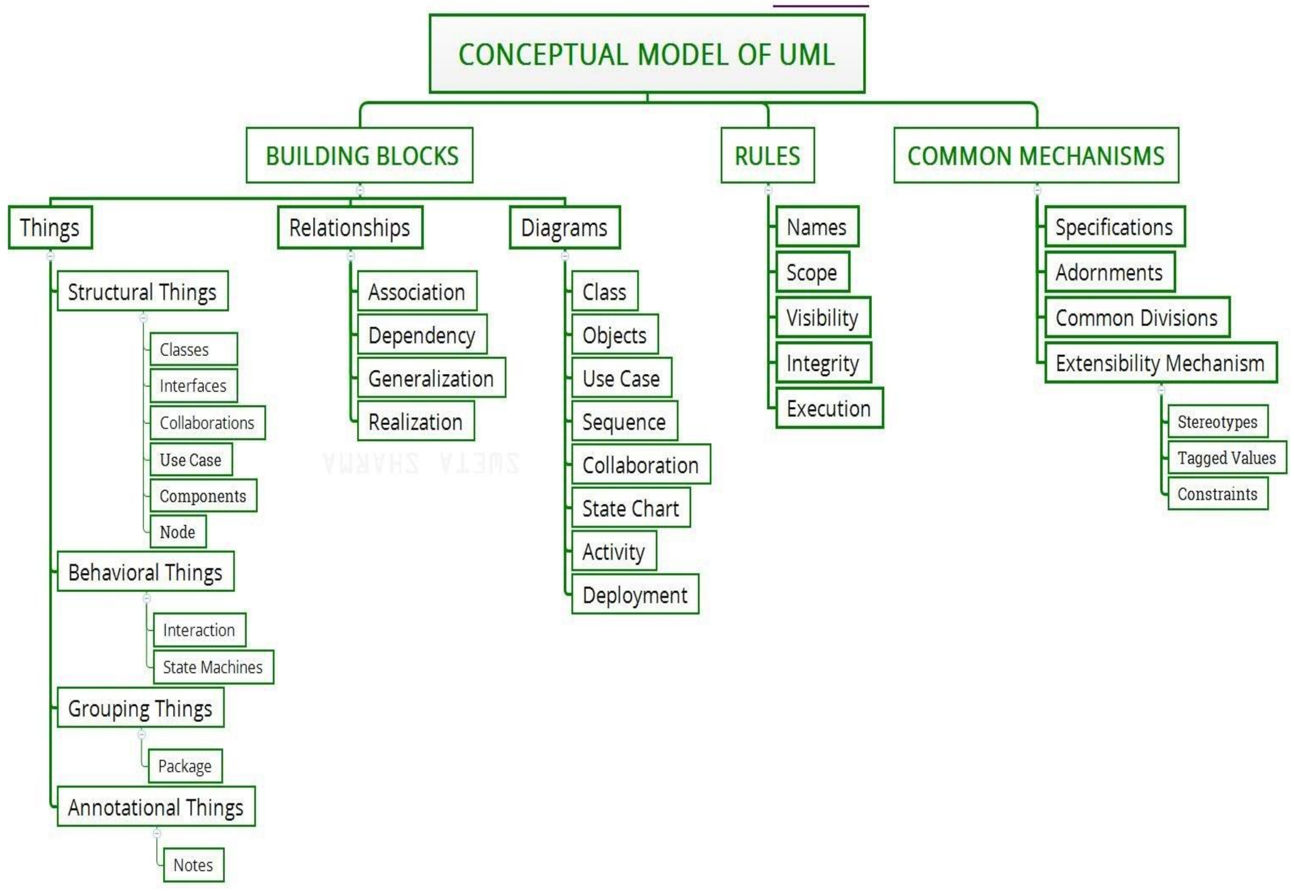


Figure – A Conceptual Model of the UML

1.Building Blocks:

The vocabulary of the UML encompasses three kinds of building blocks:

Things: Things are the abstractions that are first-class citizens in a model; relationships tie these things together; diagrams group interesting collections of things.

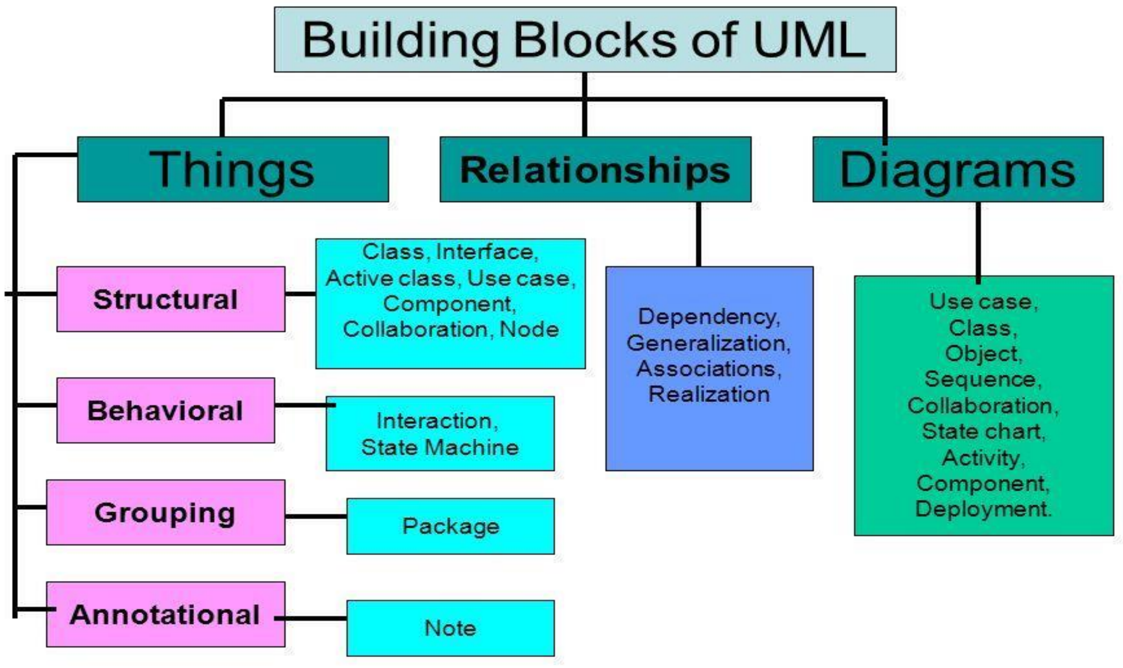
There are 4 kinds of things in the UML:

1. Structural things

2. Behavioral things

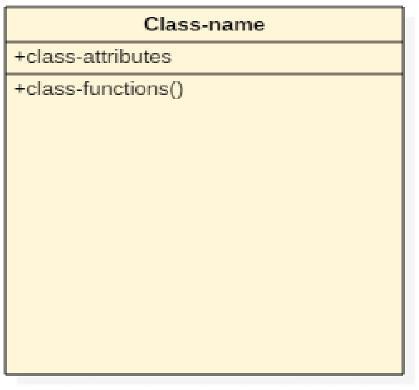
3. Grouping things

4. Annotational things


Building Blocks of UML
Things
Relationships
Diagrams
Structural
Class, Interface, Active class, Use case, Component, Collaboration, Node
Behavioral
Interaction, State Machine
Grouping
Package
Annotational
Note
Dependency, Generalization, Associations, Realization
Use case, Class, Object, Sequence, Collaboration, State chart, Activity, Component, Deployment.

A) Structural Things: Structural things are the nouns of UML models. These are the mostly static parts of a model, representing elements that are either conceptual or physical. In all, there are seven kinds of structural things.

1. Class: A class is used to represent set of objects sharing similar properties and behaviour. It is used to define the properties and operations of an object. A class whose functionalities are not defined is called an abstract class. Any UML class diagram notations are generally expressed as below UML class diagrams example,

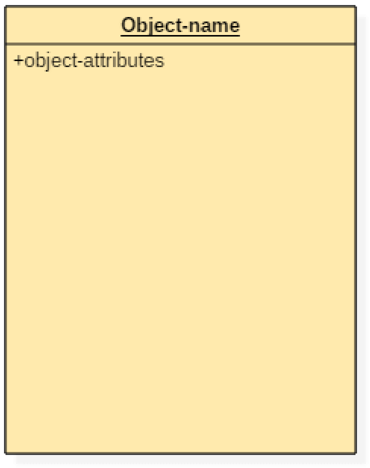


,

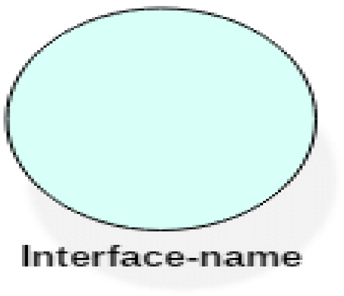
UML Class Symbol

2. Object: An object is an entity which is used to describe the behaviour and functions of a system. The class and object have the same notations. The only difference is that an object name is always underlined in UML.

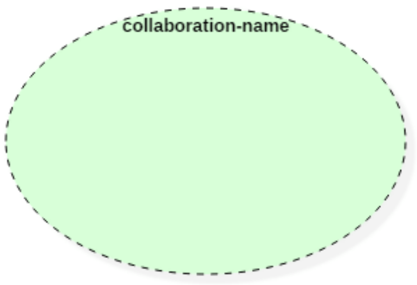
The UML notation of any object is given below.



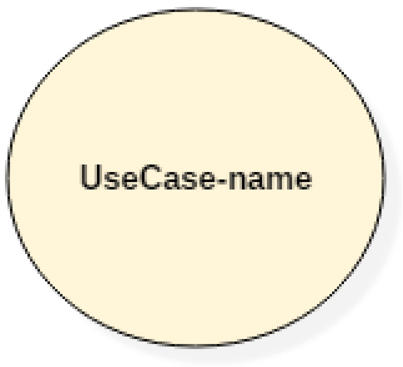
3. Interface: An interface is similar to a template without implementation details. A circle notation represents it. When a class implements an interface, its functionality is also implemented.



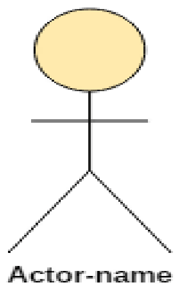
4. Collaboration: It is represented by a dotted ellipse with a name written inside it.



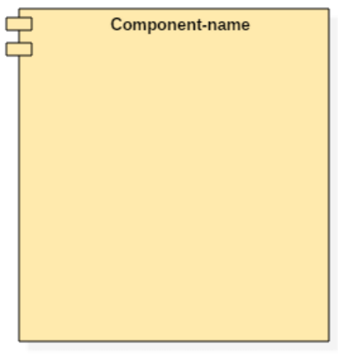
5. Use-case: Use-cases are one of the core concepts of object-oriented modelling. They are used to represent high-level functionalities and how the user will handle the system.



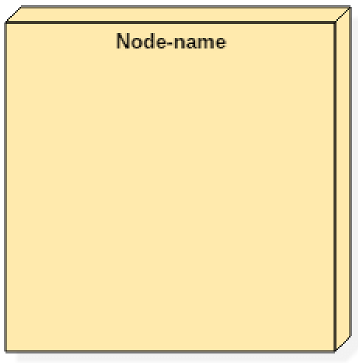
6. Actor: It is used inside use case diagrams. The Actor notation is used to denote an entity that interacts with the system. A user is the best example of an actor. The actor notation in UML is given below.



7. Component: A component notation is used to represent a part of the system. It is denoted in UML like given below,



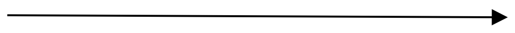
8.Node: A node is used to describe the physical part of a system. A node can be used to represent a network, server, routers, etc. Its notation is given below.



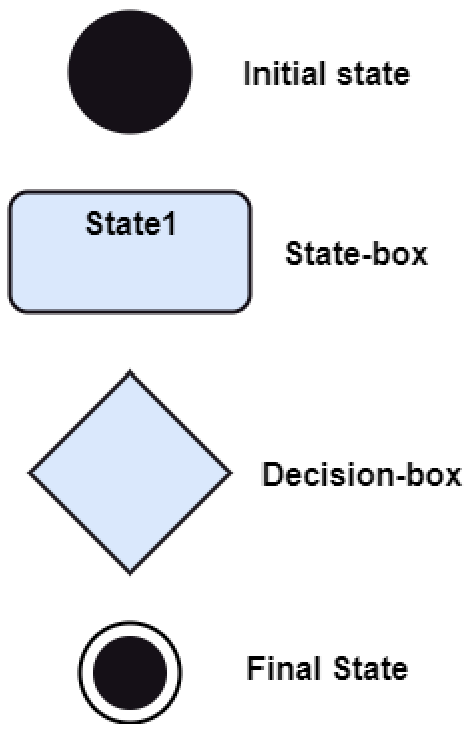
Behavioural Things:

Behavioural things are the dynamic parts of UML models. These are the verbs of a model, representing behaviour over time and space. In all, there are two primary kinds of behavioural things.

1. Messages: An interaction is a behaviour that comprises a set of messages exchanged among a set of objects within a particular context to accomplish a specific purpose. Graphically, a message is rendered as a directed line, almost always including the name of its operation display.

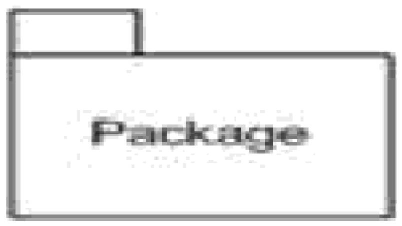


2. States: A state machine is a behaviour that specifies the sequences of states an object or an interaction goes through during its lifetime in response to events, together with its responses to those events.

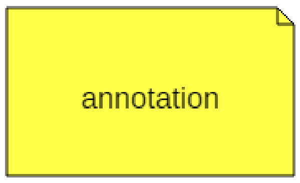


Grouping Things: Grouping things are the organizational parts of UML models. These are the boxes into which a model can be decomposed. There is one primary kind of grouping thing, namely, packages.

Packages: A package is a general-purpose mechanism for organizing elements into groups. Graphically, a package is rendered as a tabbed folder, usually including only its name and, sometimes, its contents.



Annotational Things: Anntational things are the explanatory parts of UML models. These are the comments you may apply to describe, illuminate, and remark about any element in a model. There is one primary kind of annotation thing, called a note. A note is simply a symbol for rendering constraints and comments attached to an element or a collection of elements.

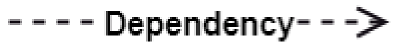


Relationships in the UML

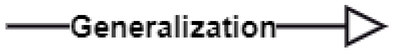
It illustrates the meaningful connections between things. It shows the association between the entities and defines the functionality of an application. There are four kinds of relationships in the UML:

1. Dependency
2. Association
3. Generalization
4. Realization

1. Dependency is a semantic relationship between two model elements in which a change to one element (the independent one) may affect the semantics of the other element (the dependent one). Graphically, a dependency is rendered as a dashed line, possibly directed, and occasionally including a label.



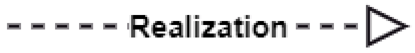
2.Association is a structural relationship among classes that describes a set of links, a link being a connection among objects that are instances of the classes. Graphically, an association is rendered as a solid line, possibly directed, occasionally including a label, and often containing other adornments, such as multiplicity and end names.



3. Generalization is a specialization/generalization relationship in which the specialized element (the child) builds on the specification of the generalized element (the parent). The child shares the structure and the behaviour of the parent. Graphically, a generalization relationship is rendered as a solid line with a hollow arrowhead pointing to the parent.

Realization

4. Realization is a semantic relationship between classifiers, wherein one classifier specifies a contract that another classifier guarantees to carry out. Generalization and a dependency relationship.



UML Diagrams

Any real-world system is used by different users. The users can be developers, testers, business people, analysts, and many more. Hence, before designing a system, the architecture is made with different perspectives in mind. The most important part is to visualize the system from the perspective of different viewers. The better we understand the better we can build the system. A diagram is the graphical presentation of a set of elements, most often rendered as a connected graph of vertices (things) and paths (relationships).

• A diagram represents an elided view of the elements that make up a system.

• In theory, a diagram may contain any combination of things and relationships.

UML plays an important role in defining different perspectives of a system. These perspectives are –

• Design

• Implementation

• Process

• Deployment

❖ The center is the Use Case view which connects all these four. A Use Case represents the functionality of the system. Hence, other perspectives are connected with use case.

❖ Design of a system consists of classes, interfaces, and collaboration. UML provides class diagram, object diagram to support this.

❖ Implementation defines the components assembled together to make a complete physical system. UML component diagram is used to support the implementation perspective.

❖ Process defines the flow of the system. Hence, the same elements as used in Design are also used to support this perspective.

❖ Deployment represents the physical nodes of the system that forms the hardware.

UML deployment diagram is used to support this perspective. The UML includes nine kinds of diagrams:

1. Class diagram
2. Object diagram
3. Use case diagram
4. Sequence diagram
5. Collaboration diagram

6. State-chart diagram

7. Activity diagram

8. Component diagram

Role of UML in Object Oriented (OO) Design

- UML is a modelling language used to model software and non-software systems.
- The emphasis is on modelling OO software applications.
- The relation between OO design and UML is very important to understand.
- The OO design is transformed into UML diagrams according to the requirement.

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.Behaviour Diagrams – Capture dynamic aspects or behaviour of thesystem. Behaviour diagrams include: Use Case Diagrams, State Diagrams,Activity Diagrams and Interaction Diagrams.The image below shows the hierarchy of diagrams according to UML

The image below shows the hierarchy of diagrams according to UML

UML- Architecture

Software architecture is all about how a software system is built at its highest level. It is needed to think big from multiple perspectives with quality and design in mind. The software team is tied to many practical concerns, such as:

• The structure of the development team.

• The needs of the business.

• The intent of the structure itself.

Software architecture provides a basic design of a complete software system. It defines the elements included in the system, the functions each element has, and how each element relates to one another. In short, it is a big picture or overall structure of the whole system, how everything works together.

To form an architecture, the software architect will take several factors into consideration:

• What will the system be used for?

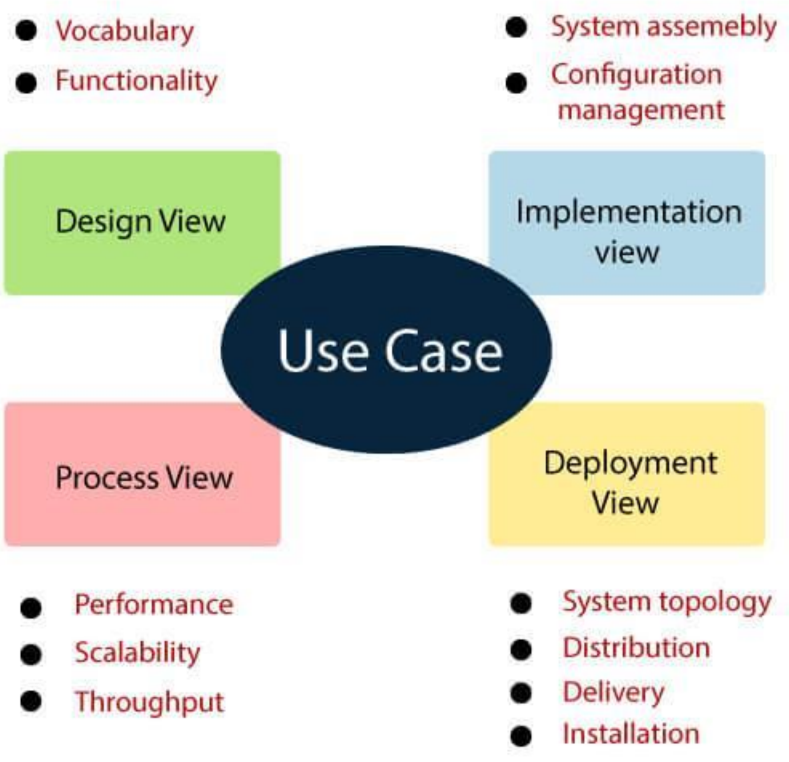
• Who will be using the system?

• What quality matters to them?

• Where will the system run?

In addition, a clear architecture will help to achieve quality in the software with a well-designed structure using principles like separation of concerns; the system becomes easier to maintain,reuse, and adapt. The software architecture is useful to people such as software developers, the project manager, the client, and the end-user. Each one will have different perspectives to view the system and will bring different agendas to a project. Also, it provides a collection of several views. It can be best understood as a collection of five views:

1. Use case view
2. Design view
3. Implementation view
4. Process view
5. Development view



Use case view

1. It is a view that shows the functionality of the system as perceived by external actors.

2. It reveals the requirements of the system.

Design View

1.It is a view that shows how the functionality is designed inside the system in terms of static structure and dynamic behavior.

2. It captures the vocabulary of the problem space and solution space.

3. With UML, it represents the static aspects of this view in class and object diagrams, whereas its dynamic aspects are captured in interaction diagrams, state chart diagrams, and activity diagrams.

Implementation View

1. It is the view that represents the organization of the core components and files.

2. It primarily addresses the configuration management of the system’s releases.

3. With UML, its static aspects are expressed in component diagrams, and the dynamic aspects are captured in interaction diagrams, state chart diagrams, and activity diagrams.

Process View

1.It is the view that demonstrates the concurrency of the system.2.
It incorporates the threads and processes that make concurrent system and synchronized mechanisms.

3. It primarily addresses the system's scalability, throughput, and performance.

4. Its static and dynamic aspects are expressed the same way as the design view but focus more on the active classes that represent these threads and processes.

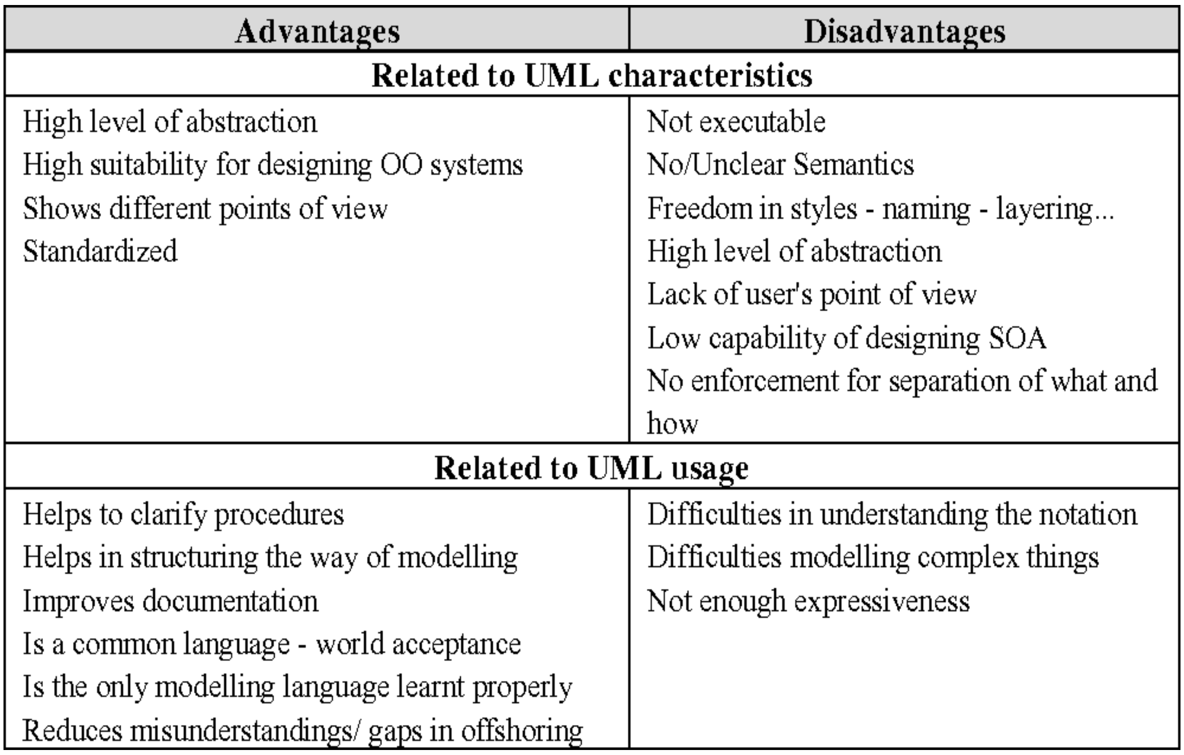
Deployment View

1. It is the view that shows the deployment of the system in terms of physical architecture.

2. It includes the nodes, which form the system hardware topology where the system will be executed.

3.It primarily addresses the distribution, delivery, and installation of the parts that build the physical system.

<u>Advantages and Disadvantages of UML:</u>



Advantages	Disadvantages
Related to UML characteristics	
High level of abstraction High suitability for designing OO systems Shows different points of view Standardized	Not executable No/Unclear Semantics Freedom in styles - naming - layering... High level of abstraction Lack of user's point of view Low capability of designing SOA No enforcement for separation of what and how
Related to UML usage	
Helps to clarify procedures Helps in structuring the way of modelling Improves documentation Is a common language - world acceptance Is the only modelling language learnt properly Reduces misunderstandings/ gaps in offshoring	Difficulties in understanding the notation Difficulties modelling complex things Not enough expressiveness

COURSE OUTCOMES of Unit 1

After the completion of the course, students should be able to:

1. Select the basic elements of modelling such as Things, Relationships and Diagrams depending on the views of UML Architecture.
2. Apply basic and Advanced Structural Modelling Concepts for designing real time applications.
3. Design Class and Object Diagrams that represent Static Aspects of a Software System.
4. Analyze Dynamic Aspects of a Software System using Use Case, Interaction and Activity Diagrams.
5. Implementation Diagrams to model behavioural aspects and Runtime environment of Software Systems.

Question Bank

TWO MARKS QUESTIONS

1. What is object-oriented modeling?
2. Define the term actor.
3. What are the models of object-oriented languages ?
4. Which technique is used to implement information hiding.
5. Define object identity.
6. What are the disadvantages of object-oriented technology ?

7. What are the advantages of object-oriented technology ?
8. Define object-oriented technology.
9. What do you mean by inheritance and polymorphism ?
10. Define the term encapsulation and abstraction.
11. What are the features of object-oriented system ?
12. What are the elements of object-oriented system ?
13. What are the benefits of object-oriented approaches ?
14. List the features of Object-oriented paradigms

FIVE MARKS QUESTIONS

1. Explain the major features of Object-Oriented Programming.
2. What is the difference between Procedure Based programming language and Object-Oriented programming language?
3. What do you understand by object identity? Explain with an example.
4. What do you mean by encapsulation? How does the object-oriented concept of message passing help to encapsulate the implementation of an object, including its data?
5. Define polymorphism. Is this concept only applicable to object-oriented systems ? Explain
6. What are the different models used in object-oriented languages?
7. What do you mean by modeling? Discuss several purposes served by models with suitable examples.
8. What are the basic principles of modeling ? Explain in detail.
9. Define link and association. Discuss the role of link and association in object modeling with suitable example
10. What do you mean by UML ? Discuss the conceptual model of UML with the help of an appropriate example
11. Describe the pros and cons of unified modeling language (UML).
12. Why UML required ? What are the basic architecture of UML ?
13. What do you understand by architectural modeling? Explain its various concepts and diagrams with suitable example.
14. Describe generalization and specialization.
15. Differentiate between a class and object with some example. Also prepare a list of objects that you would expect each of the following systems to handle :
    (1) a program for laying out a newspaper, (2) a catalog store order entry system.
16. What is a collaboration diagram ? How polymorphism is represented in a collaboration diagram? Explain with an example.
17. Explain Polymorphism, Iterated Messages and use of self in message in collaboration diagram.
18. What do you mean by sequence diagram? Explain various terms and symbols used in a sequence diagram. Describe the following using sequence diagram : (i) asynchronous messages with/without

priority. (ii) broadcast messages.

19. What do you understand by callback mechanisms ?
20. What do you mean by modeling? Discuss several purposes served by models with suitable examples.
21. Discuss the advantages of object oriented approach over structured approach.

OBJECT ORIENTED SYSTEM DESIGN

(KCS - 054)

ABES Engineering College, Ghaziabad

B. Tech Odd Semester

UNIT - 2

TOPIC:

Collaboration diagram: Terms, Concepts, depicting a message, polymorphism in collaboration 08 Diagrams, iterated messages, use of self in messages. Sequence Diagrams: Terms, concepts, depicting asynchronous messages with/without priority, call-back mechanism, broadcast messages.
Basic behavioural modelling : Use cases, Use case Diagrams, Activity Diagrams, State Machine , Process and thread, Event and signals, Time diagram, interaction diagram, Package diagram.
Architectural modelling: Component, Deployment, Component diagrams and Deployment diagrams

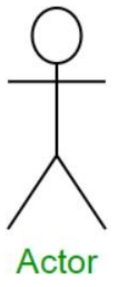
Collaboration Diagram & Sequence Diagram

Introduction

The collaboration diagram is used to show the relationship between the objects in a system. Both the sequence and the collaboration diagrams represent the same information but differently. Instead of showing the flow of messages, it depicts the architecture of the object residing in the system as it is based on object-oriented programming. The collaboration diagram, which is also known as a communication diagram,is used to portray the object's architecture in the system.

Notations used in Collaboration Diagram

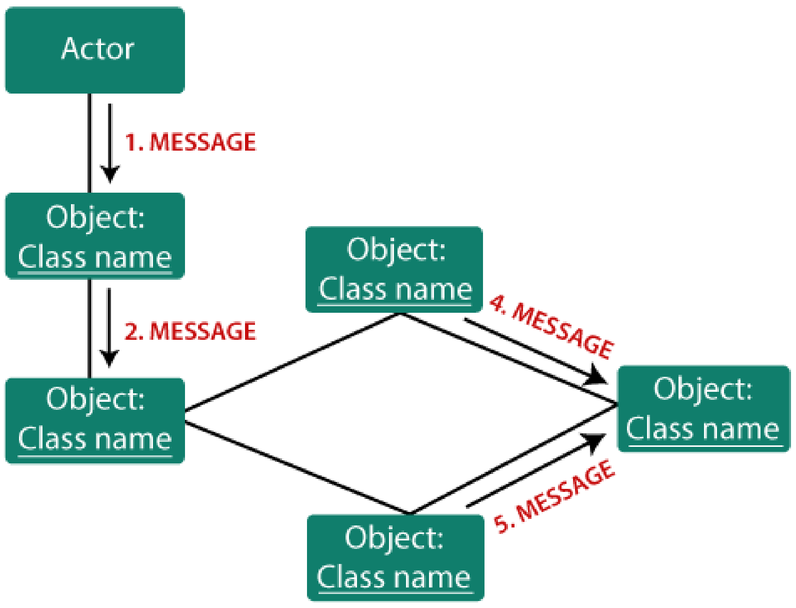
1. **Objects:** The representation of an object is done by an object symbol with its name and class underlined separated by a colon.
   Further the objects can be utilized in the following ways:
   - In the collaboration diagram, firstly, the object is created, and then its class is specified.
   - The object is represented by specifying their name and class.
   - It is not mandatory for every class to appear.
   - A class may constitute more than one object.
   - To differentiate one object from another object, it is necessary to name them.
2. **Actors:** The actor is an entity used that lies outside the scope of the system but plays the main role as it invokes the interaction. Each actor has its respective role and name. Here every actor initiates the use case. It is represented as follows:



3. **Links:** The link is an instance of association, which associates the objects and actors. It portrays a relationship between the objects through which the messages are sent. It is represented by a solid line.

4. **Messages:** It is a communication between objects which carries information and includes a sequence number, so that the activity may take place. It is represented by a labelled arrow, which is placed near a link. The messages are sent from the sender to the receiver, and the direction must be navigable in that particular direction. The receiver must understand the message.

Components of a collaboration diagram



When to use a Collaboration Diagram?

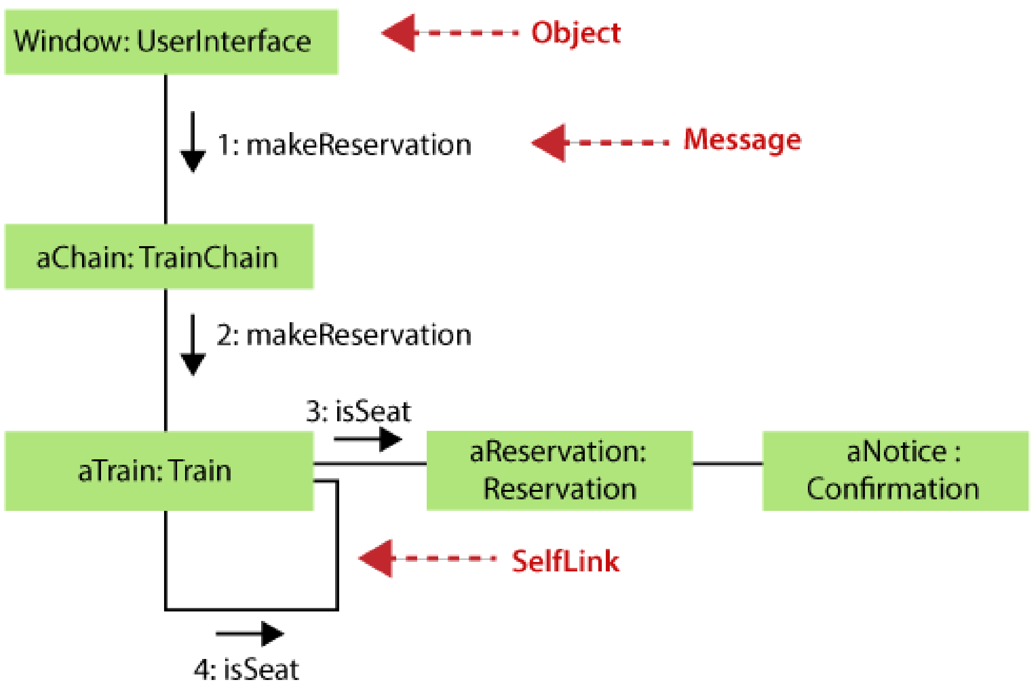
The collaborations are used when it is necessary to depict the relationship between the objects in any software domain. Both the sequence and collaboration diagrams represent the same information, but the way of portraying it quite different. The collaboration diagrams are best suited for analysing use cases.

Following are some of the use cases enlisted below for which the collaboration diagram is implemented:

1. To model collaboration among the objects or roles that carries the functionalities of use cases and operations.
2. To capture the interactions that represents the flow of messages between the objects and the roles inside the collaboration.

3. In the collaboration diagram, each message constitutes a sequence number, such that the top-level message is marked as one and so on. The messages sent during the same call are denoted with the same decimal prefix, but with different suffixes of 1, 2, etc. as per their occurrence.

Example of a Collaboration Diagram



Advantages of a Collaboration Diagram

1. It mainly puts emphasis on the structural aspect of an interaction diagram, i.e., how lifelines are connected.
2. The syntax of a collaboration diagram is similar to the sequence diagram; just the difference is that the lifeline does not consist of tails.
3. The messages transmitted over sequencing are represented by numbering each individual message.
4. The collaboration diagram is semantically weak in comparison to the sequence diagram.
5. It focuses on the elements and not the message flow, like sequence diagrams.
6. Since the collaboration diagrams are not that expensive, the sequence diagram can be directly converted to the collaboration diagram.

Disadvantages of a Collaboration Diagram

1. Multiple objects residing in the system can make a complex collaboration diagram, as it becomes quite hard to explore the objects.
2. It is a time-consuming diagram.

Sequence Diagrams in UML

The sequence diagram represents the flow of messages in the system and is also termed as an event diagram. It helps in showing several dynamic scenarios. It portrays the communication between any two lifelines as a time-ordered sequence of events, such that these lifelines took part at the run time.

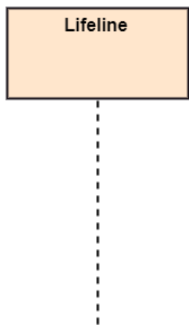
Purpose of a Sequence Diagram

1. To model high-level interaction among active objects within a system.
2. To model interaction among objects inside a collaboration realizing a use case.
3. It either models generic interactions or some certain instances of interaction.

Notations of a Sequence Diagram

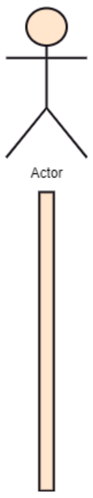
Lifeline

An individual participant in the sequence diagram is represented by a lifeline. It is positioned at the top of the diagram.



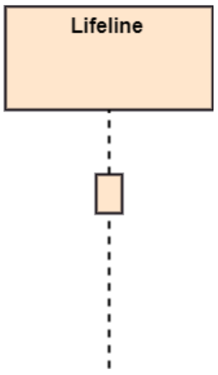
Actor

A role played by an entity that interacts with the subject is called as an actor. It is out of the scope of the system. It represents the role, which involves human users and external hardware or subjects. An actor may or may not represent a physical entity, but it purely depicts the role of an entity. Several distinct roles can be played by an actor or vice versa.



Activation

It is represented by a thin rectangle on the lifeline. It describes that time period in which an operation is performed by an element, such that the top and the bottom of the rectangle is associated with the initiation and the completion time, each respectively.

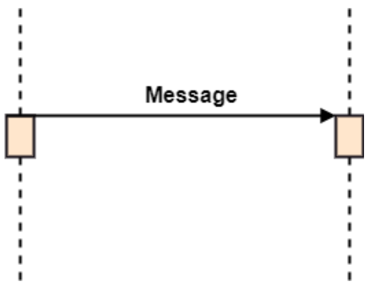


Messages

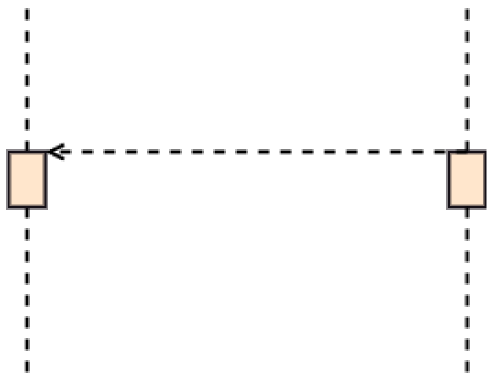
The messages depict the interaction between the objects and are represented by arrows. They are in the sequential order on the lifeline. The core of the sequence diagram is formed by messages and lifelines.

Following are types of messages enlisted below:

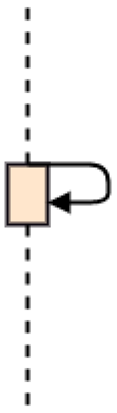
- **Call Message:** It defines a particular communication between the lifelines of an interaction, which represents that the target lifeline has invoked an operation.



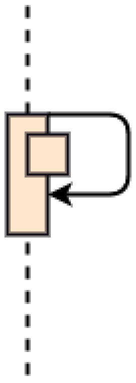
- **Return Message:** It defines a particular communication between the lifelines of interaction that represent the flow of information from the receiver of the corresponding caller message.



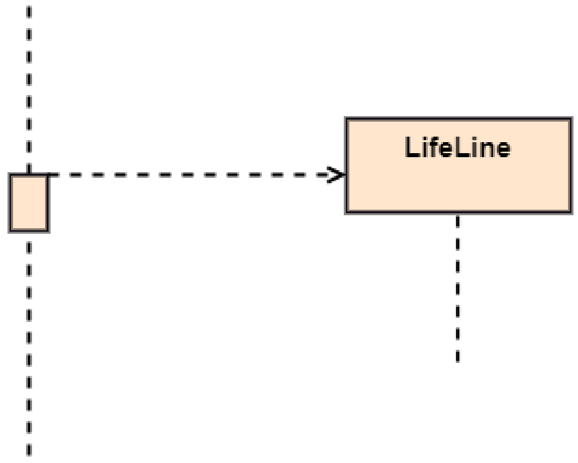
- **Self-Message:** It describes a communication, particularly between the lifelines of an interaction that represents a message of the same lifeline, has been invoked.



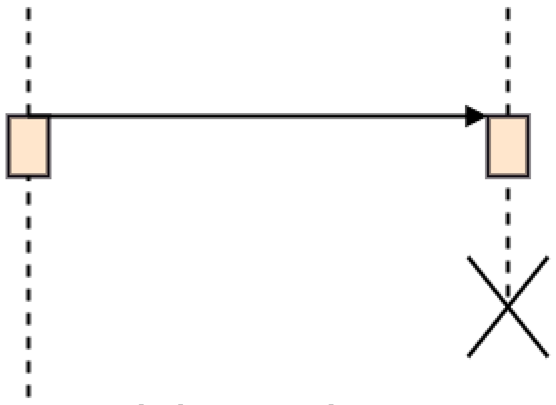
- **Recursive Message:** A self message sent for recursive purpose is called a recursive message. In other words, it can be said that the recursive message is a special case of the self message as it represents the recursive calls.



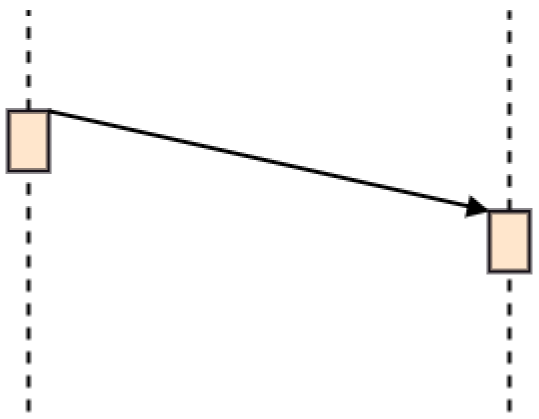
- **Create Message:** It describes a communication, particularly between the lifelines of an interaction describing that the target (lifeline) has been instantiated.



- **Destroy Message:** It describes a communication, particularly between the lifelines of an interaction that depicts a request to destroy the lifecycle of the target.
- **Duration Message:** It describes a communication particularly between the

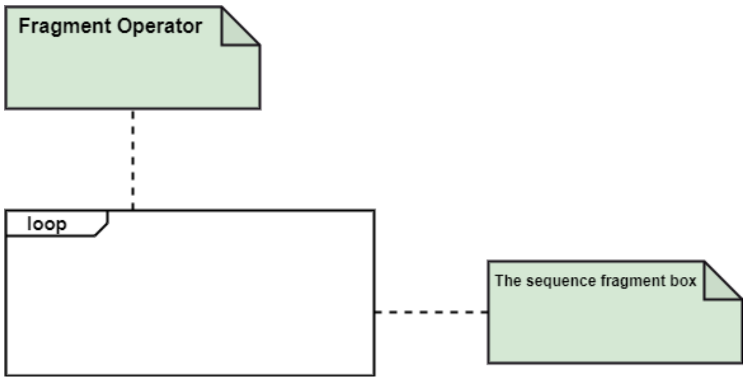


lifelines ofan interaction, which portrays the time passage of the message while modelling a system.



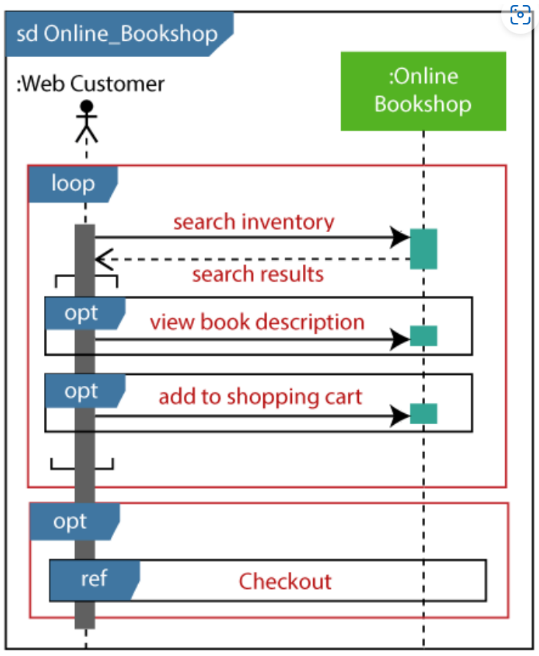
Sequence Fragments

1. Sequence fragments have been introduced by UML 2.0, which makes it quite easy forthe creation and maintenance of an accurate sequence diagram.
2. It is represented by a box called a combined fragment, encloses a part of interactioninside a sequence diagram.
3. The type of fragment is shown by a fragment operator.



Example of a Sequence Diagram

An example of a high-level sequence diagram for online bookshop is given below.
Any online customer can search for a book catalog, view a description of a particular book, add a book to its shopping cart, and do checkout.



Advantages of a Sequence Diagram

1. It explores the real-time application.
2. It depicts the message flow between the different objects.
3. It has easy maintenance.
4. It is easy to generate.
5. Implement both forward and reverse engineering.
6. It can easily update as per the new change in the system.

Disadvantages of a Sequence Diagram

a) In the case of too many lifelines, the sequence diagram can get more complex.

b) The incorrect result may be produced, if the order of the flow of messages changes.

c) Since each sequence needs distinct notations for its representation, it may make thediagram more complex.

Question Bank

TWO MARKS QUESTIONS

Collaboration DiagramBasic Terms:

i. What is the difference between sequence diagram and collaboration diagram?

ii. What is the role of actor in both kinds of interaction diagrams?

iii. Discuss the drawbacks of collaboration diagram?

FIVE MARKS QUESTIONS

i. What is the primary purpose of collaboration diagram in Behavioural modelling?

ii. Explain the role of sequence diagram in Behavioural modelling?

iii. Draw the sequence diagram of ATM Management system.

iv. Discuss different types of messages used in sequence diagrams.

v. Draw the sequence diagram of a user login registration system.

TEN MARKS QUESTIONS

i. Draw the collaboration Diagram of online book purchase system.

ii. What do we mean by a collaboration diagram? Explain various terms and symbols used in a collaboration diagram. How polymorphism is described using a collaboration diagram? Explain using an example.

AKTU QUESTIONS

1. What do we mean by a collaboration diagram? Explain various terms and symbols used in a collaboration diagram. How polymorphism is described using a collaboration diagram? Explain using an example. **(10 MARKS)**
2. Illustrate the significance of collaboration diagram and also draw a neat collaboration diagram for reserving a room in a hotel from its website. **(10 MARKS)**

Basic Behavioural Modelling

USE CASE, USE CASE DIAGRIAMS

Introduction

Use Cases: These are descriptions of how users interact with a system to achieve a specific goal. They capture the functionality and behavior of a system from an external point of view.

Use Case Diagrams: Visual representations of use cases, actors, and their relationships. Actors represent external entities (users or other systems) that interact with the system. Use cases are represented as ovals, and relationships between actors and use cases are depicted with lines.

What is the Need of Studying the Topic:

Studying Use Cases and Use Case Diagrams is essential for several reasons. They help in understanding the behavior and interactions of a system from a user's perspective. They provide aclear and structured way to document and communicate the system's functionality.

Detailed Discussion of the Topic:

Use case diagrams are one of the five diagrams in the UML for modeling the dynamic aspects of systems (activity diagrams, state chart diagrams, sequence diagrams, and collaboration diagrams are four other kinds of diagrams in the UML for modeling the dynamic aspects of systems). Use case diagrams are central to modeling the behavior of a system, a subsystem, or a class. Each one shows a set of use cases and actors andtheir relationships.

You apply use case diagrams to model the use case view of a system. For the most part, this involves modeling the context of a system, subsystem, or class, or modeling the requirements of the behavior of theseelements.

As Figure 17-1 shows, you can provide a use case diagram to model the behavior of that box• which mostpeople would call a **cellular phone**.

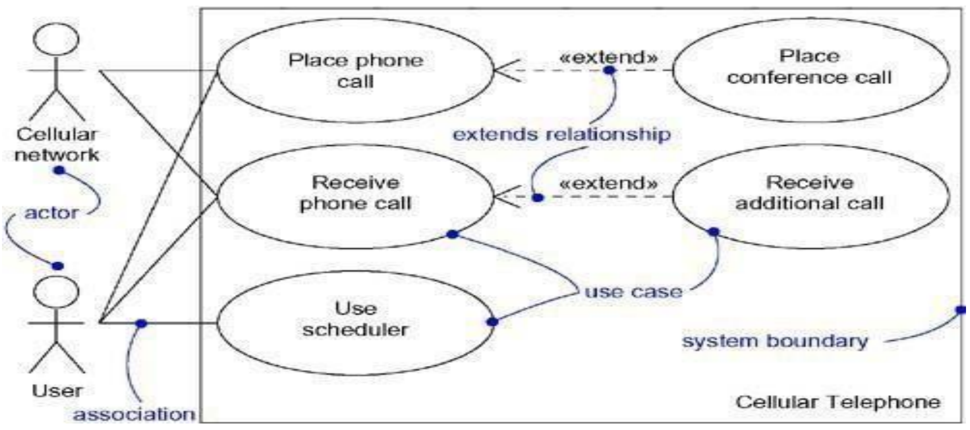


Figure 17-1 A Use Case Diagram

Terms and Concepts

A *use case diagram* is a diagram that shows a set of use cases and actors and their relationships.

Common Properties

A use case diagram is just a special kind of diagram and shares the same common properties as do all otherdiagrams• a name and graphical contents that are a projection into a model. What distinguishes a use case diagram from all other kinds of diagrams is its particular content.

Contents

Use case diagrams commonly contain

· Use cases

· Actors

· Dependency, generalization, and association relationships

Common Uses

You apply use case diagrams to model the static use case view of a system. This view primarily supports thebehavior of a system• the outwardly visible services that the system provides in the context of its environment.

When you model the static use case view of a system, you'll typically apply use case diagrams in one of twoways.

1. To model the context of a system

Modeling the context of a system involves drawing a line around the whole system and asserting which actors lie outside the system and interact with it.Here, you'll apply use case diagrams to specify the actors andthe meaning of their roles.

2. To model the requirements of a system

Modeling the requirements of a system involves specifying what that system should do (from a point of viewof outside the system), independent of how that system should do it. Here, you'll apply use case diagrams to specify the desired behavior of the system. In this manner, a use case diagram lets you view the whole systemas a black box; you can see what's outside the system and you can see how that system react to the things outside, but you can't see how that system works on the inside.

Common Modeling Techniques

1.**Modeling the Context of a System**

For example, in a credit card validation system, you'll find such things as accounts, transactions, and fraud detection agents inside the system. Similarly, you'll find such things as credit card customers and retail institutions outside the system. The things that live inside the system are responsible for carrying out the behavior that those on the outside expect the system to provide. All those things on the outside that interactwith the system constitute the system's context. This context defines the environment in which that system lives.

To model the context of a system,

Identify the actors that surround the system by considering which groups require help from the system to perform their tasks; which groups are needed to execute the system's functions; which groups interact with external hardware or other software systems; and which groups perform secondary functions foradministration and maintenance.

Organize actors that are similar to one another in a generalization/specialization hierarchy.

Where it aids understandability, provide a stereotype for each such actor.

Populate a use case diagram with these actors and specify the paths of communication from each actor tothe system's use cases.

For example, Figure 17-2 shows the context of a credit card validation system, with an emphasis on the actorsthat surround the system.

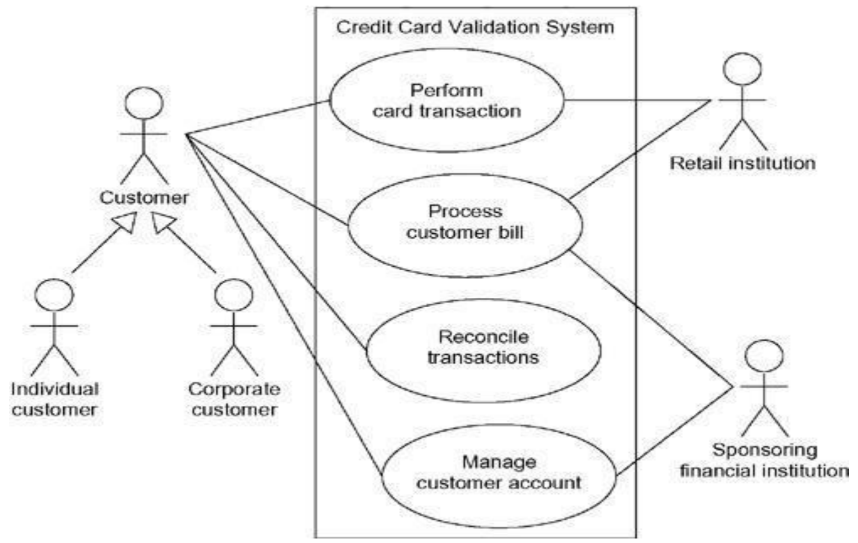


Figure 17-2 Modeling the Context of a System

2. Modeling the Requirements of a System

A requirement is a design feature, property, or behavior of a system. When you state a system's requirements, you are asserting a contract, established between those things that lie outside the system and the system itself, which declares what you expect that system to do. For the most part, you don't care how the system does it, you just care *that* it does it. A well-behaved system will carry out all its requirements faithfully, predictably, and reliably. When you build a system, it's important to start with agreement about what that system should do, although you will certainly evolve your understanding of those requirements as you iteratively and incrementally implement the system.

To model the requirements of a system,

- Establish the context of the system by identifying the actors that surround it.
- For each actor, consider the behavior that each expects or requires the system to provide.
- Name these common behaviors as use cases.
- Factor common behavior into new use cases that are used by others; factor variant behavior into new usecases that extend more main line flows.
- Model these use cases, actors, and their relationships in a use case diagram.

· Adorn these use cases with notes that assert nonfunctional requirements; you may have to attach some ofthese to the whole system.

3. Forward and Reverse Engineering

Most of the UML's other diagrams, including class, component, and state chart diagrams, are clear candidatesfor forward and reverse engineering because each has an analog in the executable system. Use case diagramsare a bit different in that they reflect rather than specify the implementation of a system, subsystem, or class. Use cases describe how an element behaves, not how that behavior is implemented, so it cannot be directly forward or reverse engineered.

To forward engineer a use case diagram,

· For each use case in the diagram, identify its flow of events and its exceptional flow of events.

· Depending on how deeply you choose to test, generate a test script for each flow, using the flow'spreconditions as the test's initial state and its postconditions as its success criteria.

· As necessary, generate test scaffolding to represent each actor that interacts with the use case. Actors that push information to the element or are acted on by the element may either be simulated or substituted by itsreal-world equivalent.

· Use tools to run these tests each time you release the element to which the use case diagram applies. To reverse engineer a use case diagram,

· Identify each actor that interacts with the system.

· For each actor, consider the manner in which that actor interacts with the system, changes the state of thesystem or its environment, or responds to some event.

· Trace the flow of events in the executable system relative to each actor. Start with primary flows and only laterconsider alternative paths.

· Cluster related flows by declaring a corresponding use case. Consider modeling variants using extend relationships, and consider modeling common flows by applying include relationships.

· Render these actors and use cases in a use case diagram, and establish their relationships.

- Use Cases and Use Case Diagrams are fundamental concepts in software engineering and systemdesign. They are part of the Unified Modeling Language (UML) and are used to describe the functional requirements of a software system.

Characteristics Related to the Topic:

- Use Cases focus on user interactions and system functionality.
- Use Case Diagrams emphasize the relationships between actors and use cases.

Advantages & Disadvantages:

Advantages:

- Improve communication between stakeholders as they provide a visual and clearrepresentation of system functionality.
- Aid in requirements gathering and validation.
- Help in identifying missing or redundant features.

Disadvantages:

- May not cover every detail of system behavior, especially non-functional requirements.
- Can become complex for large and intricate systems.

Outcomes of the Topic Taught:

- Learning about Use Cases and Use Case Diagrams enables students and professionals to effectively gather and communicate system requirements, enhancing the quality and clarity of software development processes.

Potential Applications & Implementation in Industries:

Use Cases and Use Case Diagrams are widely used in the software industry to document and communicate requirements. They also find applications in system engineering and business process modeling. Industries such as healthcare, finance, and manufacturing benefit from their use in systemdesign and development.

ACTIVITY DIAGRAMS

Introduction of the Topic:

Activity Diagrams are a fundamental part of the Unified Modeling Language (UML) used in software engineering and systems design. They provide a visual representation of the workflow within a system or process, showing how various activities and actions are interconnected.

- Activity Diagrams:
  - These diagrams represent the dynamic aspects of a system or process.
  - They include actions, control nodes, and transitions.
  - Actions represent specific operations or work to be performed.
  - Control nodes, such as initial and final nodes, decision nodes, and merge nodes, control theflow of activities.
  - Transitions are directed lines connecting activities and control nodes, showing the order ofexecution.

What is the Need of Studying the Topic:

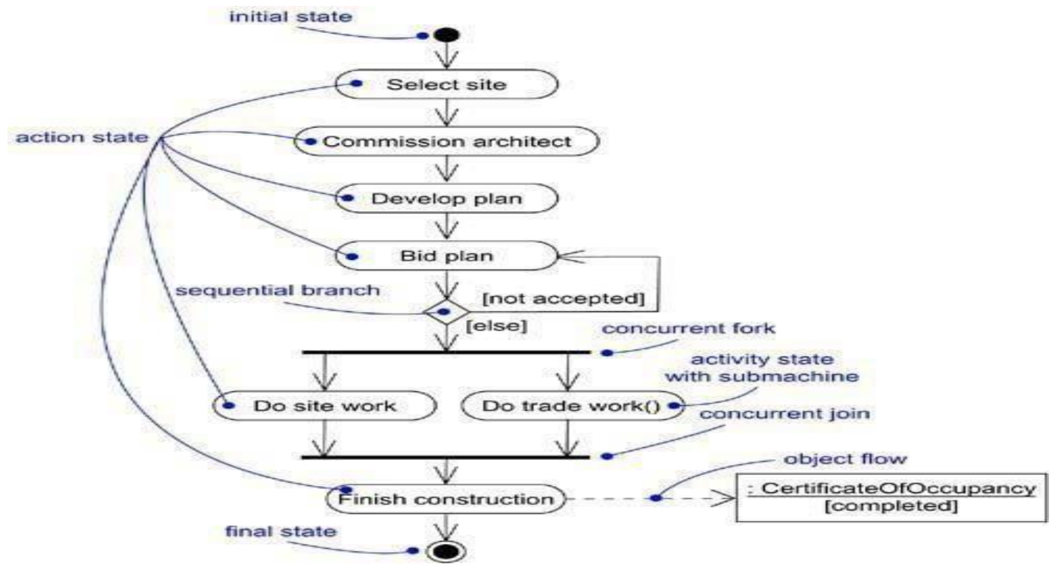
Studying Activity Diagrams is crucial for several reasons. They help in modeling, understanding, and visualizing the workflow of a system or process. This aids in effective communication, analysis,and design of systems, leading to more efficient development.

Detailed Discussion of the Topic:

Activity diagrams are one of the five diagrams in the UML for modeling the dynamic aspects of systems. An activity diagram is essentially a flowchart, showing flow of control from activity to activity. You use activity diagrams to model the dynamic aspects of a system. For the most part, this involves modeling the sequential (and possibly concurrent) steps in a computational process. Consider the workflow associated withbuilding a house.

An activity is an ongoing nonatomic execution within a state machine. Activities ultimately result in some action, which is made up of executable atomic computations that results in a change in state of the system orthe return of a value.

Activity diagrams are not only important for modeling the dynamic aspects of a system, but



also forconstructing executable systems through forward and reverse engineering.

Terms and Concepts

An *activity diagram* shows the flow from activity to activity. An is an ongoing nonatomic execution within astate machine. Activities ultimately result in some *action*, which is made up of executable atomic computations that result in a change in state of the system or the return of a value. Graphically, an activity diagram is a collection of vertices and arcs.

Common Properties

An activity diagram is just a special kind of diagram and shares the same common properties as do all other diagrams• a name and graphical contents that are a projection into a model. What distinguishes an interactiondiagram from all other kinds of diagrams is its content.

Contents: Activity diagrams commonly contain

· Activity states and action states

· Transitions

· Objects

Action States and Activity States

In the flow of control modeled by an activity diagram, things happen. You might evaluate some expressionthat sets the value of an attribute or that returns some value. Alternately, you might call an operation on an object, send a signal to an object, or even create or destroy an object.

These executable, atomic computations are called action states because they are states of the system, each representing the execution of an action. As Figure 19-2 shows, you represent an action state using a lozenge shape (a symbol with horizontal top and bottom and convex sides). Inside that shape, you may write any expression.

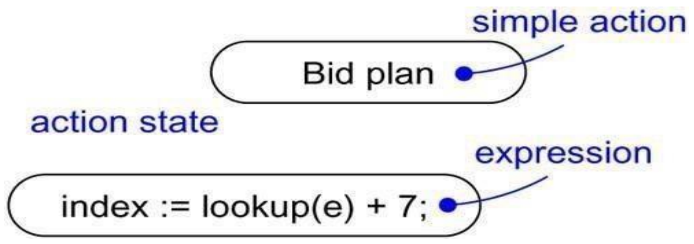


Figure 19-2 Action States

Action states can't be decomposed. Furthermore, action states are atomic, meaning that events may occur, butthe work of the action state is not interrupted. Finally, the work of an action state is generally considered to take insignificant execution time.

An action state is an activity state that cannot be further decomposed. Similarly, you can think of an activity state as a composite, whose flow of control is made up of other activity states and action states. Zoom into thedetails of an activity state, and you'll find another activity diagram. As Figure 19-3 shows, there's no notational distinction between action and activity states, except that an activity state may have additional parts, such as entry and exit actions (actions which are involved on entering and leaving the state, respectively) and submachine specifications.

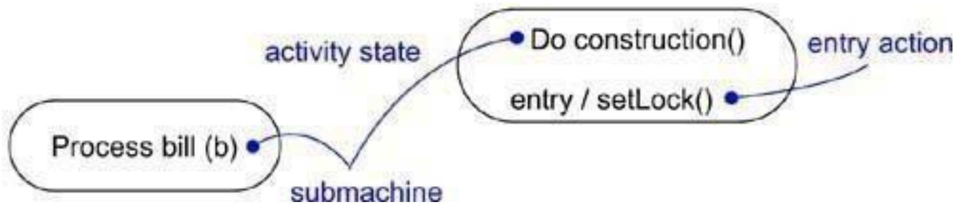


Figure 19-3 Activity States

Action states and activity states are just special kinds of states in a state machine. When you enter an actionor activity state, you simply perform the action or the activity; when you finish, control passes to the next action or activity. Activity states are somewhat of a shorthand, therefore. An activity state is semantically equivalent to expanding its activity graph (and transitively so) in place until you only see actions.

Transitions

When the action or activity of a state completes, flow of control passes immediately to the next action or activity state. You specify this flow by using transitions to show the path from one action or activity state to the next action or activity state. In the UML, you represent a transition as a simple directed line, as Figure 19-4 shows.

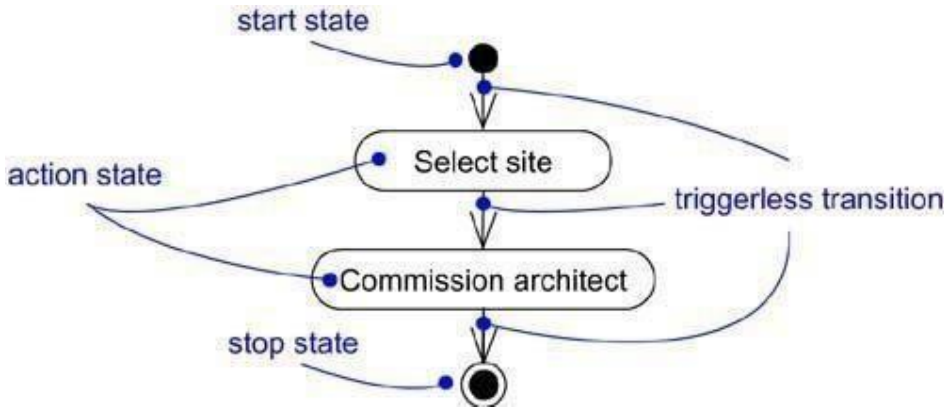


Figure 19-4 Triggerless Transitions

Semantically, these are called triggerless, or completion, transitions because control passes immediately oncethe work of the source state is done.

Branching

Simple, sequential transitions are common, but they aren't the only kind of path you'll need to model a flowof control. As in a flowchart, you can include a branch, which specifies alternate paths taken based on some Boolean expression. As Figure 19-5 shows, you represent a branch as a diamond. A branch may have one incoming transition and two or

more outgoing ones. On each outgoing transition, you place a Boolean expression, which is evaluated only once on entering the branch. Across all these outgoing transitions, guards should not overlap (otherwise, theflow of control would be ambiguous), but they should cover all possibilities (otherwise, the flow of control would freeze).

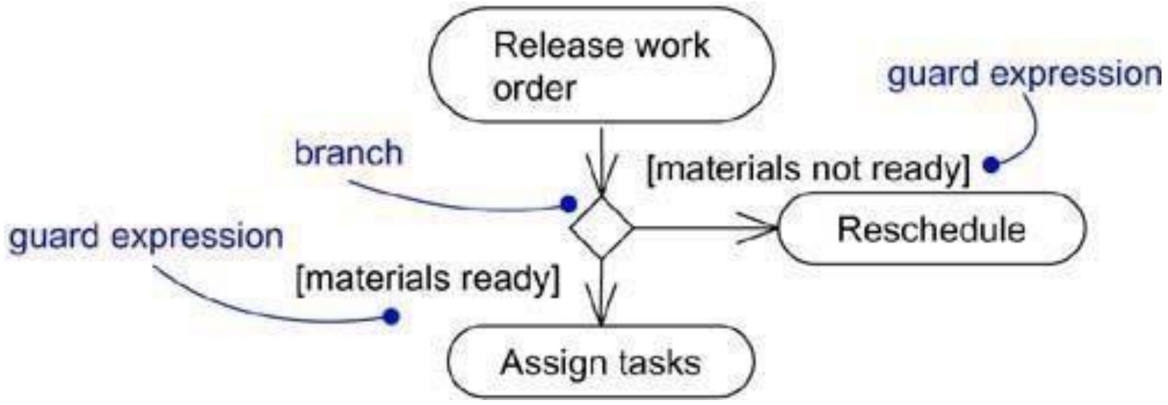


Figure 19-5 Branching

Forking and Joining

Simple and branching sequential transitions are the most common paths you'll find in activity diagrams. However• especially when you are modeling workflows of business processes• you might encounter flows that are concurrent. In the UML, you use a synchronization bar to specify the forking and joining of these parallel flows of control. A synchronization bar is rendered as a thick horizontal or vertical line.

For example, consider the concurrent flows involved in controlling an audio-animatronic device that mimics human speech and gestures.

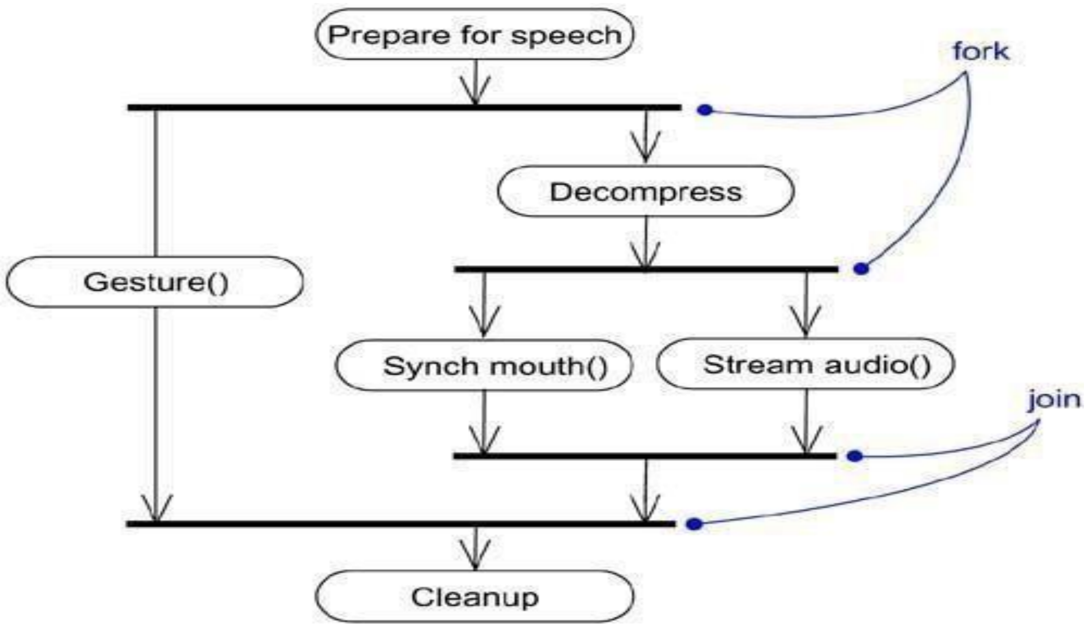


Figure 19-6 Forking and Joining Swim lanes

You'll find it useful, especially when you are modeling workflows of business processes, to partition the activity states on an activity diagram into groups, each group representing the business organization responsible for those activities. In the UML, each group is called a swimlane because, visually, each group isdivided from its neighbor by a vertical solid line, as shown in Figure 19-7. A swimlane specifies a locus of activities.

There's a loose connection between swimlanes and concurrent flows of control. Conceptually, the activities of each swimlane are generally• but not always• considered separate from the activities of neighboring swimlanes. That makes sense because, in the real world, the business organizations that generally map to these swimlanes are independent and concurrent.

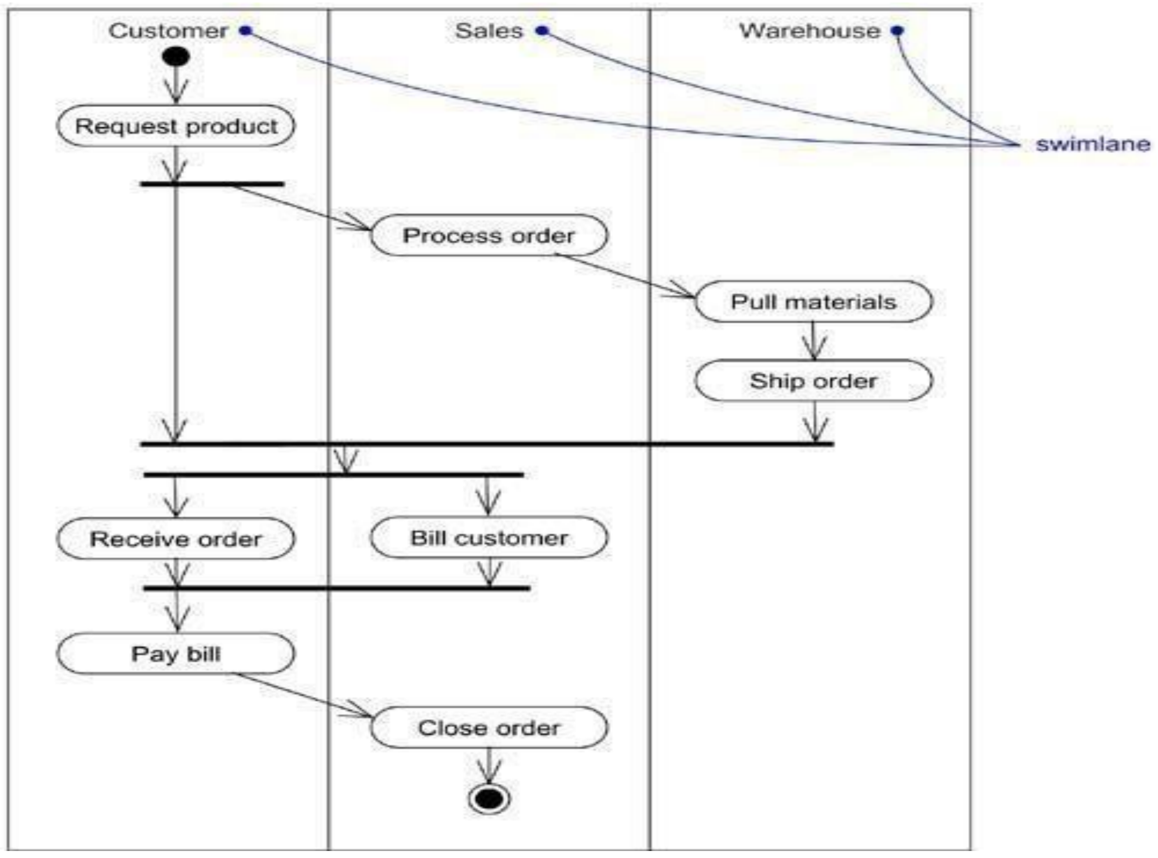


Figure 19-7 Swimlanes

Object Flow

Objects may be involved in the flow of control associated with an activity diagram. For example, in the workflow of processing an orderas in the previous figure, the vocabulary of your problem space will also include such classes as Order and Bill. Instances of these two classes will be produced bycertain activities(Process order will create an Order object, for example); other activities may modify these objects (for example, Ship order will change the state of the Order object to filled).

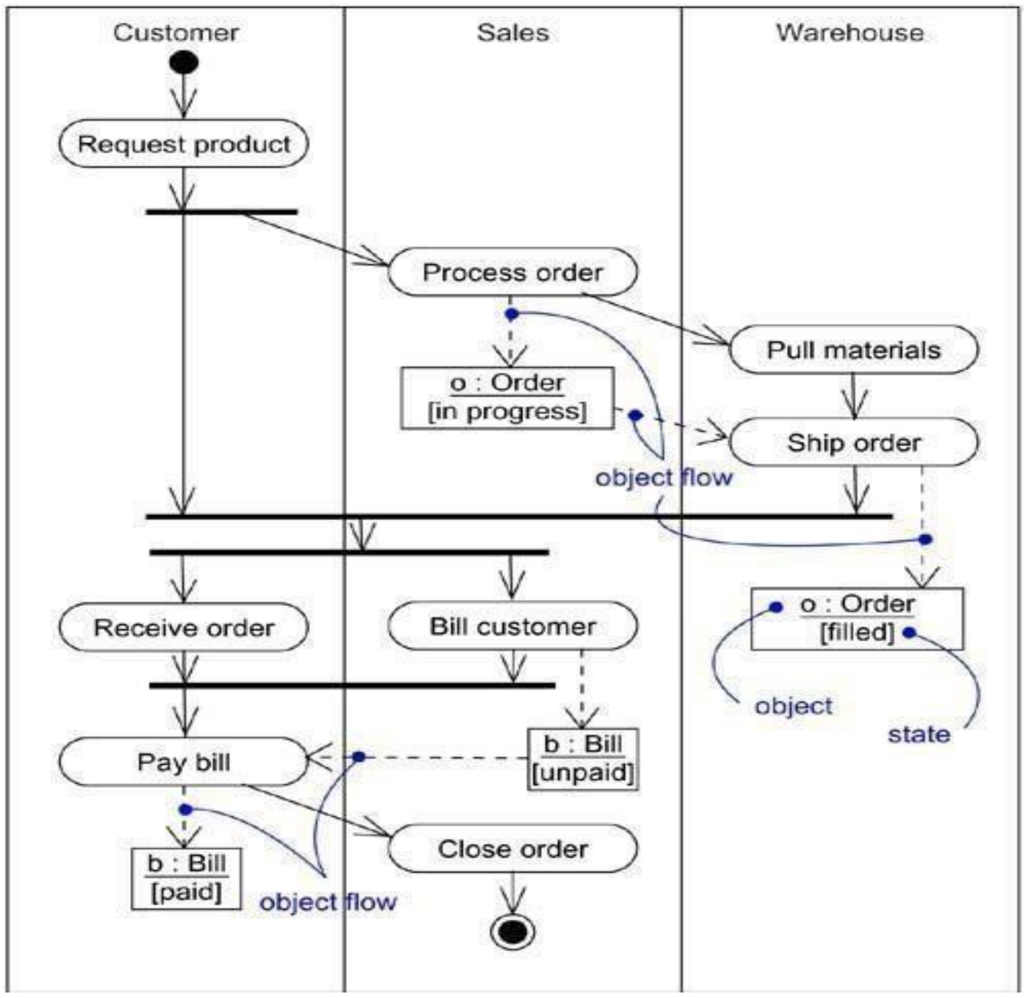


Figure 19-8 Object Flow

Common Uses

You use activity diagrams to model the dynamic aspects of a system. These dynamic aspects may involve theactivity of any kind of abstraction in any view of a system's architecture, including classes (which includes active classes), interfaces, components, and nodes.

When you model the dynamic aspects of a system, you'll typically use activity diagrams in two ways.

1. **To model a workflow**

Here you'll focus on activities as viewed by the actors that collaborate with the system. Workflows often lieon the fringe of software-intensive systems and are used to visualize, specify, construct, and document business processes that involve the system you are developing. In this use of activity diagrams, modeling object flow is particularly important.

2. **To model an operation**

Here you'll use activity diagrams as flowcharts, to model the details of a computation. In this use of activity diagrams, the modeling of branch, fork, and join states is particularly important. The context of an activitydiagram used in this way involves the parameters of the operation and its local objects.

3. **Common Modeling Techniques 1. Modeling a Workflow**

No software-intensive system exists in isolation; there's always some context in which a system lives, and that context always encompasses actors that interact with the system. You can model the business processesfor the way these various automated and human systems collaborate by using activity diagrams.

To model a workflow,

Establish a focus for the workflow. For nontrivial systems, it's impossible to show all interesting workflowsin one diagram. Select the business objects that have the high-level responsibilities for parts of the overall workflow. These may be real things from the vocabulary of the system, or they may be more abstract. In either case, create a swimlane for each important business object.

Identify the preconditions of the workflow's initial state and the postconditions of the workflow's final state.This is important in helping you model the boundaries of the workflow.

Beginning at the workflow's initial state, specify the activities and actions that take place over time andrender them in the activity diagram as either activity states or action states.

For complicated actions, or for sets of actions that appear multiple times, collapse these into activity states,and provide a separate activity diagram that expands on each.

Render the transitions that connect these activity and action states. Start with the sequential flows in theworkflow first, next consider branching, and only then consider forking and joining.

If there are important objects that are involved in the workflow, render them in the activity diagram, aswell. Show their changing values and state as necessary to communicate the intent of the object flow. For example, Figure 19-9 shows an activity diagram for a retail business, which specifies the workflowinvolved when a customer returns an item from a mail order.

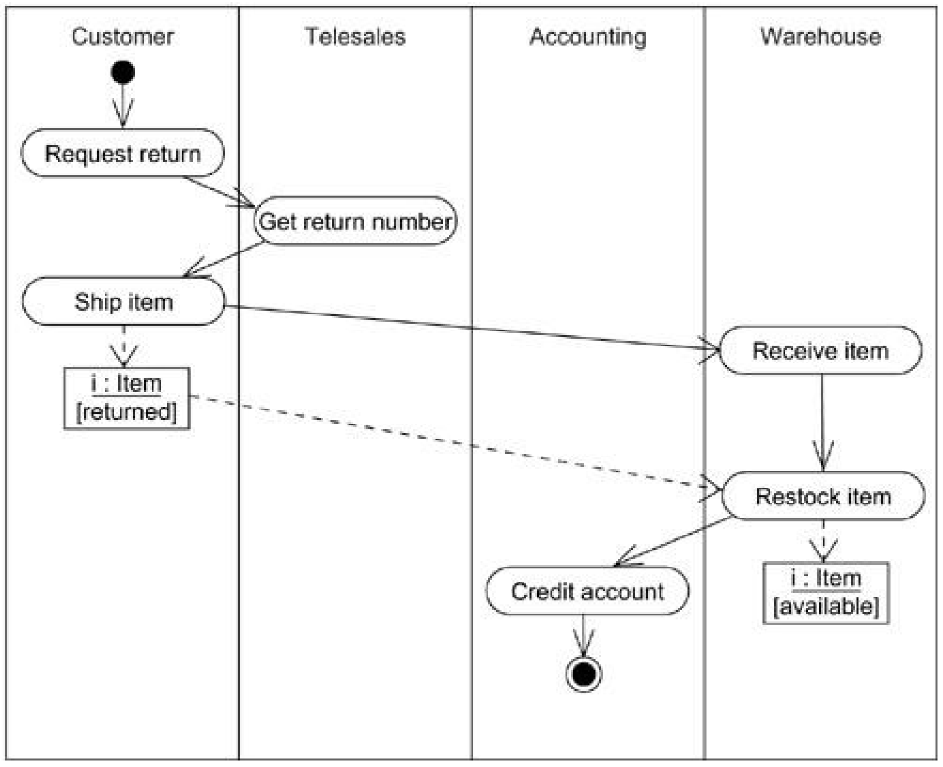


Figure 19-9 Modeling a Workflow

2. Modeling an Operation

The most common element to which you'll attach an activity diagram is an operation. Used in this manner, an activity diagram is simply a flowchart of an operation's actions. An activity diagram's primary advantage is thatall the elements in the diagram are semantically tied to a rich underlying model. For example, any other operation or signal that an action state references can be type checked against the class of the target object.

To model an operation,

- Collect the abstractions that are involved in this operation. This includes the operation's parameters (including its return type, if any), the attributes of the enclosing class, and certain neighboring classes.
- Identify the preconditions at the operation's initial state and the postconditions at the operation's final state. Also identify any invariants of the enclosing class that must hold during the execution of the operation.

· Beginning at the operation's initial state, specify the activities and actions that take place over time andrender them in the activity diagram as either activity states or action states.

· Use branching as necessary to specify conditional paths and iteration.

· Only if this operation is owned by an active class, use forking and joining as necessary to specify parallelflows of control.

Forward and Reverse Engineering

Forward engineering (the creation of code from a model) is possible for activity diagrams, especially if thecontext of the diagram is an operation. For example, using the previous activity diagram, a forward engineering tool could generate the following C++ code for the operation intersection.

```
Point Line::intersection (l : Line)

{
if (slope == l.slope)
return Point(0,0);
int x = (l.delta - delta) / (slope - l.slope);
int y = (slope * x) + delta;
return Point(x, y);
}
```

There's a bit of cleverness here, involving the declaration of the two local variables? A less sophisticated toolmight have first declared the two variables and then set their values.

Reverse engineering (the creation of a model from code) is also possible for activity diagrams, especially ifthe context of the code is the body of an operation. In particular, the previous diagram could have been generated from the implementation of the class Line.

(ii) Flowchart/Algorithm/Models:

- Activity Diagrams are a visual modeling technique and do not involve detailed algorithms. They aresimilar to flowcharts in that they represent the flow of activities within a process.

Characteristics Related to the Topic:

- Activity Diagrams emphasize the flow of actions within a system or process.
- They focus on the sequence of actions, decision points, parallel activities, and synchronization.

Advantages & Disadvantages:

- Advantages:
    - Provide a clear and visual representation of workflow.
    - Improve communication among stakeholders.
    - Help in identifying inefficiencies, bottlenecks, and opportunities for optimization.
- Disadvantages:
    - Can become complex for large and intricate processes.
    - May not capture every detail of the system behavior.

Outcomes of the Topic Taught:

Learning about Activity Diagrams equips individuals to model, analyze, and design the dynamic aspects of systems and processes. This leads to more efficient and organized workflows.

Potential Applications & Implementation in Industries:

Activity Diagrams are widely used in various industries, including software development, manufacturing, healthcare, and business process management. They are used to document and optimize workflows, resulting in increased productivity and reduced errors in processes.

State Machine , Process and thread, Event and signals

Introduction of the Topic:

- A state machine is a conceptual model used in software engineering to represent the behavior of an object or a system. It's characterized by a finite set of states, transitions between these states, and theconditions that trigger these transitions.

What is the Need of Studying the Topic:

- Understanding state machines is essential for modeling and controlling the behavior of complex systems, such as software applications, embedded systems, and even hardware devices. State machines help in making the behavior of systems more predictable and manageable.

Detailed Discussion of the Topic:

- **State Machine:**
  - Consists of states, transitions, and events.
  - States represent different conditions or phases of an object or system.
  - Transitions define how an object or system moves between states.
  - Events trigger these transitions.

In the real world, things happen. Not only do things happen, but lots of things may happen at the same time, and at the most unexpected times. "Things that happen" are called events, and each one represents the specification of a significant occurrence that has a location in time and space. In the context of state machines, you use events to model the occurrence of a stimulus that can trigger a state transition. Events mayinclude signals, calls, the passing of time, or a change in state.

Events may be synchronous or asynchronous, so modeling events is wrapped up in the modelling ofprocesses and threads.

In the UML, each thing that happens is modeled as an event. The UML provides a graphical representation of an event, as Figure 20-1 shows. This notation permits you to visualize the declaration of events (such as the signal OffHook), as well as the use of events to trigger a state transition (such as the signal Off Hook, which causes a transition from the Active to the

Idle state of a telephone).

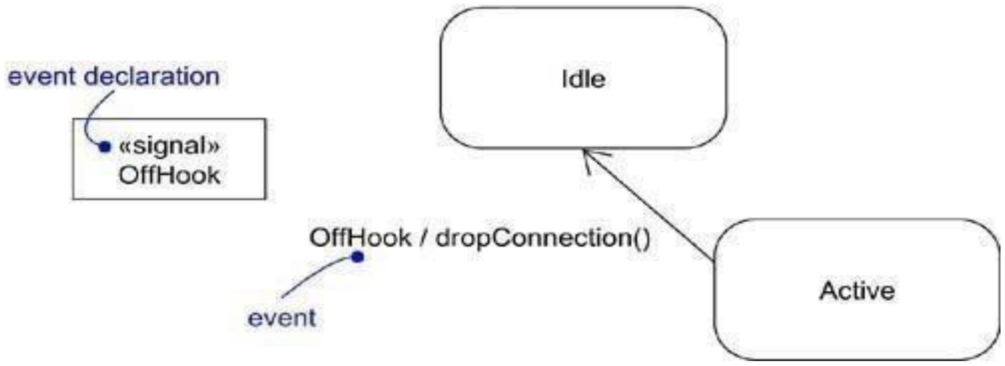


Figure 20-1 Events

Terms and Concepts

An *event* is the specification of a significant occurrence that has a location in time and space. In the context of state machines, an event is an occurrence of a stimulus that can trigger a state transition. A *signal* is a kind of event that represents the specification of an asynchronous stimulus communicated between instances.

Kinds of Events

Events may be external or internal. External events are those that pass between the system and its actors. For example, the pushing of a button and an interrupt from a collision sensor are both examples of external events. Internal events are those that pass among the objects that live inside the system. An overflow exception is an example of an internal event.

In the UML, you can model four kinds of events: signals, calls, the passing of time, and a change in state.

Signals

A signal represents a named object that is dispatched (thrown) asynchronously by one object and then received (caught) by another. Exceptions are supported by most contemporary programming languages andare the most common kind of internal signal that you will need to model.

A signal may be sent as the action of a state transition in a state machine or the sending of a message in an interaction. The execution of an operation can also send signals. In fact, when you model a class or an interface, an important part of specifying the behavior of that element is specifying the signals that its operations can send. In the UML, you model the relationship between an operation and the events that it cansend by using a dependency relationship, stereotyped as send.

In the UML, as Figure 20-2 shows, you model signals (and exceptions) as stereotyped classes. You can use adependency, stereotyped as send, to indicate that an operation sends a particular signal.

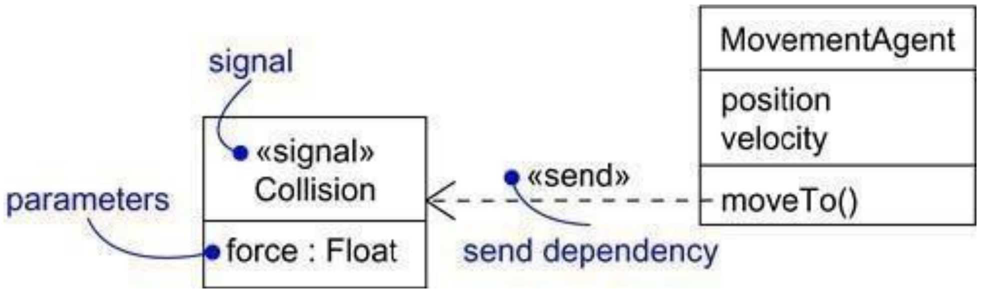


Figure 20-2 Signals

Call Events

Just as a signal event represents the occurrence of a signal, a call event represents the dispatch of an operation. In both cases, the event may trigger a state transition in a state machine. Whereas a signal is anasynchronous event, a call event is, in general, synchronous. This means that when an object invokes an operation on another object that has a state machine, control passes from the sender to the receiver, the transition is triggered by the event, the operation is completed, the receiver transitions to a new state, andcontrol returns to the sender.

As Figure 20-3 shows, modeling a call event is indistinguishable from modeling a signal event.

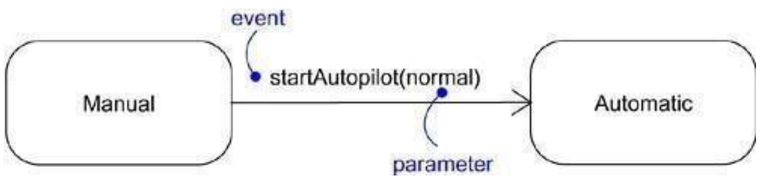


Figure 20-3 Call Events

Time and Change Events

A time event is an event that represents the passage of time. As Figure 20-4 shows, in the UML you model atime event by using the keyword after followed by some expression that evaluates to a period of time. Such expressions can be simple (for example, after 2 seconds) or complex (for example, after 1 ms since exiting Idle). Unless you specify it explicitly, the starting time of such an expression is the time since entering the current state.

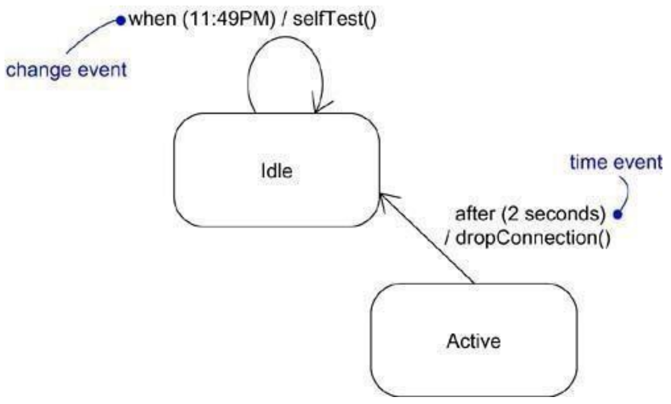


Figure 20-4 Time and Change Events

3. State Machines

Using a state machine, you can model the behavior of an individual object. A state machine is a behavior that specifies the sequences of states an object goes through during its lifetime in response to events, together withits responses to those events.

You use state machines to model the dynamic aspects of a system. For the most part, this involves specifyingthe lifetime of the instances of a class, a use case, or an entire system. An activity is an ongoing nonatomic execution within a state machine. Activities ultimately result in some action, which is made up of executableatomic computations that result in a change in state of the model or a return of a value. The state of an objectis a condition or situation during the life of an object during which it satisfies some condition, performs some activity, or waits for some event.

You can visualize a state machine in two ways: by emphasizing the flow of control from activity to activity(using activity diagrams), or by emphasizing the potential states of the objects and the transitions among those states (using state chart diagrams). You use state machines to model the behavior of any modeling element, although, most commonly, that will be a class, a use case, or an entire system. State machines may be visualized in two ways. First, using activitydiagrams, you can focus on the activities that take place within the object. Second, using state chart diagrams,you can focus on the event ordered behavior of an object, which is especially useful in modeling reactive systems.

The UML provides a graphical representation of states, transitions, events, and actions, as Figure 21-1 shows.This notation permits you to visualize the behavior of an object in a way that lets you emphasize the important elements in the life of that object.

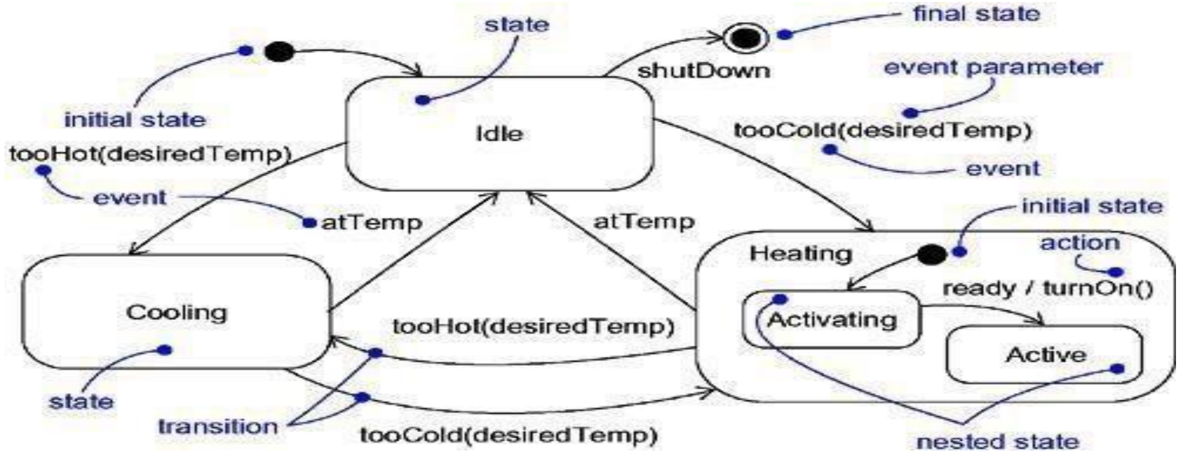


Figure 21-1 State Machines

Terms and Concepts

A *state machine* is a behavior that specifies the sequences of states an object goes through during its lifetime in response to events, together with its responses to those events. A *state* is a condition or situation during thelife of an object during which it satisfies some condition, performs some activity, or waits for some event. An

event is the specification of a significant occurrence that has a location in time and space. In the context of state machines, an event is an occurrence of a stimulus that can trigger a state transition. A *transition* is a relationship between two states indicating that an object in the first state will perform certain actions and enter the second state when a specified event occurs and specified conditions are satisfied. An *activity* is ongoing nonatomic execution within a state machine. An *action* is an executable atomic computation that results in a change in state of the model or the return of a value. Graphically, a state is rendered as a rectanglewith rounded corners. A transition is rendered as a solid directed line.

Context

Every object has a lifetime. On creation, an object is born; on destruction, an object ceases to exist. In between, an object may act on other objects (by sending them messages), as well as be acted on (by being thetarget of a message). In many cases, these messages will be simple, synchronous operation calls. For example, an instance of the class Customer might invoke the operation getAccountBalance on an instance of the class BankAccount. Objects such as these don't need a state machine to specify their behavior because their current behavior does not depend on their past.

States

A state is a condition or situation during the life of an object during which it satisfies some condition, performs some activity, or waits for some event. An object remains in a state for a finite amount of time. Forexample, a Heater in a home might be in any of four states: Idle (waiting for a command to start heating the house), Activating (its gas is on, but it's waiting to come up to temperature), Active (its gas and blower are both on), and Shutting Down (its gas is off but its blower is on, flushing residual heat from the system).

As Figure 21-2 shows, you represent a state as a rectangle with rounded corners.

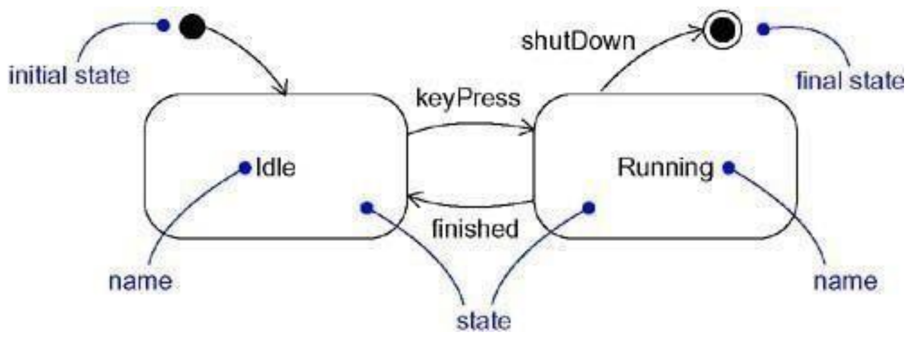


Figure 21-2 States

Initial and Final States

As the figure shows, there are two special states that may be defined for an object's state machine. First, there's the initial state, which indicates the default starting place for the state machine or substate. An initial state is represented as a filled black circle. Second, there's the final state, which indicates that the execution ofthe state machine or the enclosing state has been completed. A final state is represented as a filled black circle surrounded by an unfilled circle.

Initial and final states are really pseudo states. Neither may have the usual parts of anormal state, except for a name. A transition from an initial state to a final state may have the full complement of features, including aguard condition and action (but not a trigger event).

Transitions

A transition is a relationship between two states indicating that an object in the first state will perform certainactions and enter the second state when a specified event occurs and specified conditions are satisfied. On such a change of state, the transition is said to fire. Until the transition fires, the object is said to be in the source state; after it fires, it is said to be in the target state. For example, a Heater might transition from the Idle to the Activating state when an event such as too Cold (with the parameter desired Temp) occurs.

As Figure 21-3 shows, a transition is rendered as a solid directed line from the source to the target state. Aself-transition is a transition whose source and target states are the same.

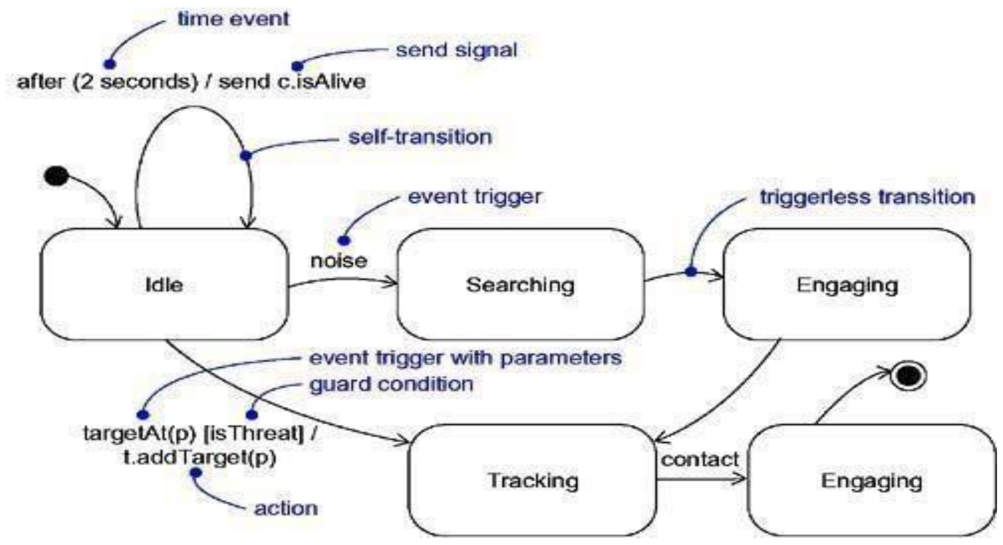


Figure 21-3 Transitions

Event Trigger

An event is the specification of a significant occurrence that has a location in time and space. In the contextof state machines, an event is an occurrence of a stimulus that can trigger a state transition. As shown in theprevious figure, events may include signals, calls, the passing of time, or a change in state. A signal or a callmay have parameters whose values are available to the transition, including expressions for the guard condition and action.

It is also possible to have a triggerless transition, represented by a transition with no event trigger. A triggerless transition• also called a completion transition• is triggered implicitly when its source state hascompleted its activity. An event trigger may be polymorphic. For example, if you've specified a family ofsignals, then a transition whose trigger event is S can be triggered by S, as well as by any children of S.

Guard

A guard condition is rendered as a Boolean expression enclosed in square brackets and placed after the trigger event. A guard condition is evaluated only after the trigger event for its transition occurs. Therefore, it's possible to have multiple transitions from the same source state and with the same event trigger, as long as those conditions don't overlap.

Action

An action is an executable atomic computation. Actions may include operation calls (to the object that owns the state machine, as well as to other visible objects), the creation or destruction of another object, or the sending of a signal to an object. An action is atomic, meaning that it cannot be interrupted by an event and therefore runs to completion. This is in contrast to an activity, which may be interrupted by other events.

Entry and Exit Action

In a number of modeling situations, you'll want to dispatch the same action whenever you enter a state, no matter which transition led you there. Similarly, when you leave a state, you'll want to dispatch the same action no matter which transition led you away. For example, in a missile guidance system, you might want to explicitly announce the system is on Track whenever it's in the Tracking state, and off Track whenever it's out of the state. Using flat state machines, you can achieve this effect by putting those actions on every entering and exiting transition, as appropriate. However, that's somewhat error prone; you have to rememberto add these actions every time you add a new transition. Furthermore, modifying this action means that youhave to touch every neighboring transition.
Entry and exit actions may not have arguments or guard conditions. However, the entry action at the top levelof a state machine for a class may have parameters that represent the arguments that the machine receives when the object is created.

Internal Transitions

Once inside a state, you'll encounter events you'll want to handle without leaving the state. These are calledinternal transitions, and they are subtly different from self-transitions.

Internal transitions may have events with parameters and guard conditions. As such, internal transitions areessentially interrupts.

Activities

When an object is in a state, it generally sits idle, waiting for an event to occur. Sometimes, however, you may wish to model an ongoing activity. While in a state, the object does some work that will continue until itis interrupted by an event. For example, if an object is in the Tracking state, it might follow Target as long asit is in that state.

Processes and Threads

In the UML, you model each independent flow of control as an active object that represents a process or thread that can initiate control activity. A process is a heavyweight flow that can execute concurrently with other processes; a thread is a lightweight flow that can execute concurrently with other threads within the same process. Building abstractions so that they work safely in the presence of multiple flows of control is hard. In particular, you have to consider approaches to communication and synchronization that are more complex than for sequential systems. You also have to be careful to neither over-engineer your process view (too many concurrent flows and your system ends up thrashing) nor under engineer it (insufficient concurrency does not optimize the system's throughput.

In the UML, each independent flow of control is modeled as an active object. An active object is a process orthread that can initiate control activity. As for every kind of object, an activeobject isaninstanceof aclass. In thiscase, anactive object isaninstanceof anactive class. Also as for every kind of object, active objects can communicate with one another by passing messages, although here, message passing must be extended with certain concurrency semantics, to help you to synchronize the interactions among independent flows.

The UML provides a graphical representation of an active class, as Figure 22-1 shows. Active classes are kinds of classes, so have all the usual compartments for class name, attributes, and operations. Active classes often receive signals, which you typically enumerate in an extra compartment.

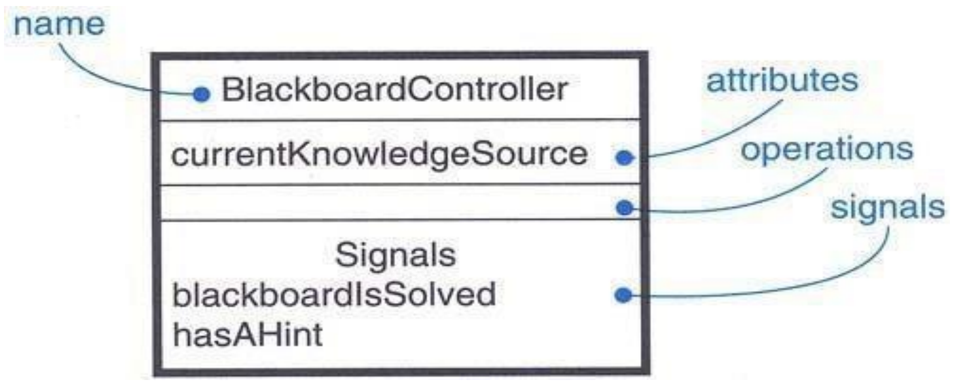


Figure 22-1 Active Class

Terms and Concepts

An active object is an object that owns a process or thread and can initiate control activity. An active class is a class whose instances are active objects. A process is a heavyweight flow that can execute concurrently with other processes. A thread is a lightweight flow that can execute concurrently with other threads within the same process. Graphically, an active class is rendered as a rectangle with thick lines. Processes and threads are rendered as stereotyped active classes (and also appear as sequences in interaction diagrams).

Flow of Control

In a purely sequential system, there is one flow of control. This means that one thing, and one thing only, cantake place at a time. When a sequential program starts, control is rooted at the beginning of the program and operations are dispatched one after another. Even if there are concurrent things happening among the actors outside the system, a sequential program will process only one event at a time, queuing or discarding any concurrent external events.

In the UML, you use an active class to represent a process or thread that is the root of an independent flow ofcontrol and that is concurrent with all peer flows of control.

You can achieve true concurrency in one of three ways: first, by distributing active objects across multiplenodes; second, by placing active objects on nodes with multiple processors; and third, by a combination ofboth methods.

Classes and Events

Active classes are just classes, albeit ones with a very special property. An active class represents an independent flow of control, whereas a plain class embodies no such flow.

In contrast to active classes, plain classes are implicitly called passive because they cannot independently initiate control activity.

Speaking of state machines, both passive and active objects may send and receive signal events and call events. the caller waits for the receiver to accept the call; the operation is invoked; a return object (if any) is passed back to the caller; and then the two continue on their independent paths. For the duration of the call, the two flows of controls are in lock step. Second, one active object might asynchronously send a signal or call an operation of another object. That kind of communication has mailbox semantics, which means that the caller sends the signal or calls the operation and then continues on its independent way. In the meantime, the receiver accepts the signal or call whenever it is ready (with intervening events or calls queued) and continues on its way after it is done. This is called a mailbox because the two objects are not synchronized; rather, one object drops off a message for the other.

In the UML, you render a synchronous message as a full arrow and an asynchronous message as a half arrow, as in Figure 22-2.

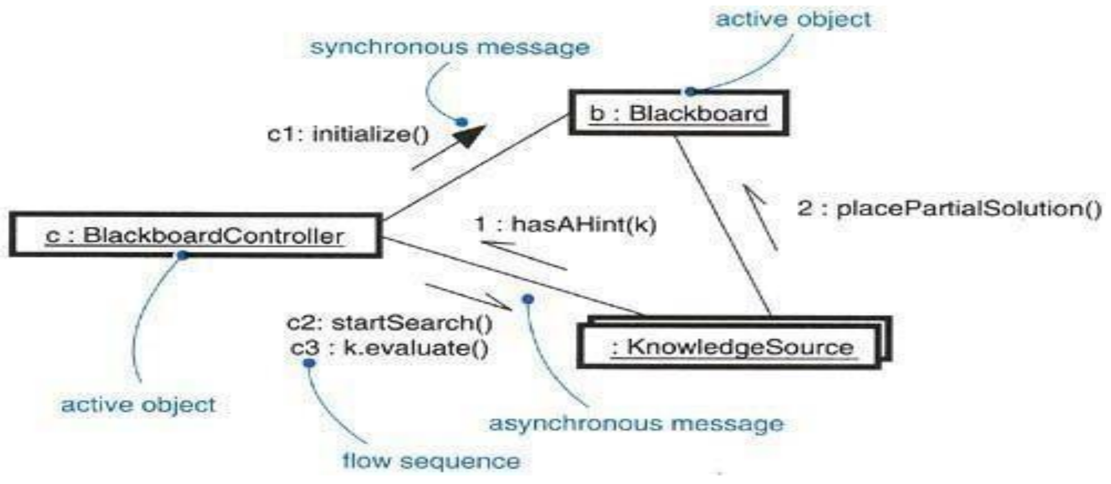


Figure 22-2 Communication

Third, a message may be passed from an active object to a passive object. A difficulty arises if more than one active object at a time passes their flow of control through one passive object. In that situation, you have to model the synchronization of these two flows very carefully, as discussed in the next section.

Fourth, a message may be passed from a passive object to an active one. At first glance, this may seem illegal, but if you remember that every flow of control is rooted in some active object, you'll understand that a passive object passing a message to an active object has the same semantics as an active object passing a message to an active object.

It is possible to model variations of synchronous and asynchronous message passing by using constraints. For example, to model a balking rendezvous as found in Ada, you'd use a synchronous message with a constraint such as {wait = 0}, saying that the caller will not wait for the receiver. Similarly, you can model a time out by using a constraint such as {wait = 1 ms}, saying that the caller will wait no more than one millisecond for the receiver to accept the message.

State chart Diagrams

State chart diagrams are one of the five diagrams in the UML for modeling the dynamic aspects of systems. A statechart diagram shows a state machine. An activity diagram is a special case of a statechart diagram in which all or most of the states are activity states and all or most of the transitions are triggered by completion of activities in the source state. Thus, both activity and statechart diagrams are useful in modeling the lifetime of an object. However, whereas an activity diagram shows flow of control from activity to activity, a statechart diagram shows flow of control from state to state.

In the UML, you model the event-ordered behavior of an object by using statechart diagrams. As Figure 24-1 shows, a statechart diagram is simply a presentation of a state machine, emphasizing the flow of control from state to state.

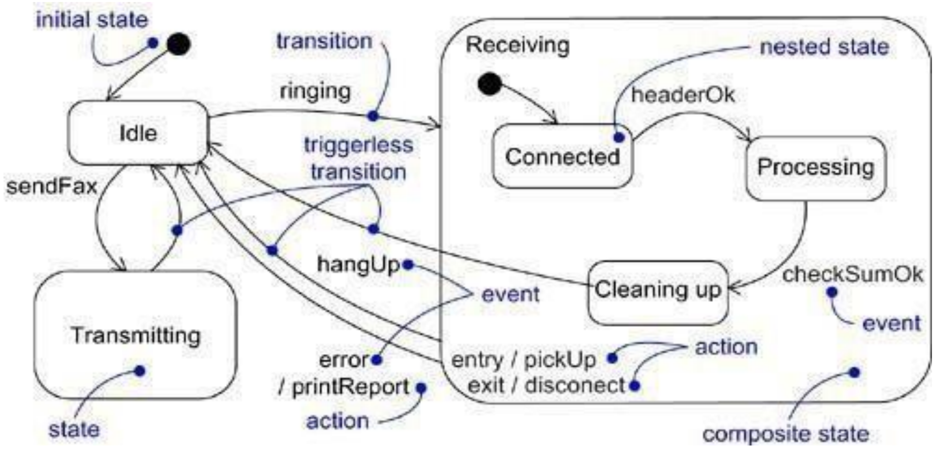


Figure 24-1 Statechart Diagram

Terms and Concepts

A *statechart diagram* shows a state machine, emphasizing the flow of control from state to state. A *state machine*is a behavior that specifies the sequences of states an object goes through during its lifetime in response to events, together with its responses to those events. A *state* is a condition or situation in the life of an object during which it satisfies some condition, performs some activity, or waits for some event. An *event* is the specification of a significant occurrence that has a location in time and space. Inthecontext of state machines, an event is an occurrence of a stimulus that can trigger a state transition. A *transition* is a relationship between two states indicating that an object in the first state will perform certain actions and enter the second state when a specified event occurs and specified conditions are satisfied. An *activity* is ongoing nonatomic execution within a state machine. An *action* is an executable atomic computation that results in a change in state of the model or the return of a value. Graphically, a statechart diagram is a collection of vertices and arcs.

Common Properties

A statechart diagram is just a special kind of diagram and shares the same common properties as do all otherdiagrams• that is, a name and graphical contents that are a projection into a model. What distinguishes a statechart diagram from all other kinds of diagrams is its content.

Contents

Statechart diagrams commonly contain

· Simple states and composite states

· Transitions, including events and actions

Common Uses

You use statechart diagrams to model the dynamic aspects of a system. These dynamic aspects may involve theevent-ordered behavior of any kind of object in any view of a system's architecture, including classes (which includes active classes), interfaces, components, and nodes.

When you model the dynamic aspects of a system, a class, or a use case, you'll typically use statechart diagramsin one way.

· To model reactive objects

Common Modeling Technique Modeling Reactive Objects To model a reactive object,

· Choose the context for the state machine, whether it is a class, a use case, or the system as a whole.

- Choose the initial and final states for the object. To guide the rest of your model, possibly state the pre- andpostconditions of the initial and final states, respectively.
- Decide on the stable states of the object by considering the conditions in which the object may exist for someidentifiable period of time. Start with the high-level states of the object and only then consider its possible substates.
- Decide on the meaningful partial ordering of stable states over the lifetime of the object.
- Decide on the events that may trigger a transition from state to state. Model these events as triggers totransitions that move from one legal ordering of states to another.
- Attach actions to these transitions (as in a Mealy machine) and/or to these states (as in a Moore machine).
- Consider ways to simplify your machine by using substates, branches, forks, joins, and history states.
- Check that all states are reachable under some combination of events.
- Check that no state is a dead end from which no combination of events will transition the object out of thatstate.
- Trace through the state machine, either manually or by using tools, to check it against expected sequences ofevents and their responses.

For example, Figure 24-2 shows the statechart diagram for parsing a simple context-free language, such as youmight find in systems that stream in or stream out messages to XML. In this case, the machine is designed to parsea stream of characters that match the syntax

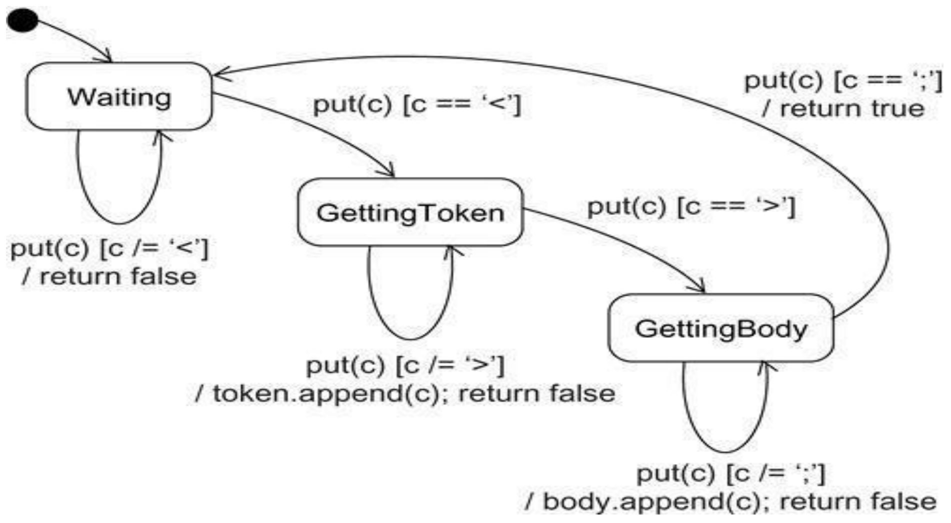


Figure 24-2 Modeling Reactive Objects

Characteristics Related to the Topic:

- State machines provide a clear and structured way to model the behavior of systems.
- They can be hierarchical, meaning that states can contain sub-states, allowing for complex behaviormodeling.

Advantages & Disadvantages:

- Advantages:
    - Improve the clarity and predictability of system behavior.
    - Facilitate design, testing, and debugging of complex systems.
- Disadvantages:
    - May become cumbersome for very large systems.
    - Requires careful attention to detail in defining states and transitions.

Outcomes of the Topic Taught:

- Learning about state machines equips individuals with a powerful tool for modeling and controlling system behavior. This leads to more reliable and robust system designs.

Potential Applications & Implementation in Industries:

- State machines find applications in various industries, including software development, robotics, andmanufacturing. They are used to model and control the behavior of complex systems and processes.

<u>Time Diagram</u>

Introduction of the Topic:

- A time diagram is a graphical representation used to visualize and analyze the timing and sequencing ofevents, processes, or signals over time. It's a valuable tool in various fields, including engineering, electronics, and software development.

What is the Need of Studying the Topic:

- Studying time diagrams is crucial for understanding and managing the temporal aspects of complexsystems, which is essential in ensuring that processes or events occur in the correct order and withinspecific time constraints.

Detailed Discussion of the Topic:

- Time Diagram:
    - Represents time on one axis and events, processes, or signals on the other.
    - Events are typically marked as points or vertical lines along the time axis.
    - Arrows or lines connect events to illustrate their temporal relationships.

Flowchart/Algorithm/Models:

- Time diagrams are a visual representation and do not involve specific flowcharts, algorithms, or models. They serve to provide a graphical view of the timing relationships within a system.

Characteristics Related to the Topic:

- Time diagrams are used to:
    - Visualize the timing and sequencing of events.
    - Identify potential timing conflicts or errors.
    - Analyze the behavior of systems or processes over time.

Advantages & Disadvantages:

- Advantages:
    - Improve the understanding of temporal relationships in complex systems.
    - Aid in identifying and resolving timing issues.

    - Facilitate communication and collaboration among team members.

- Disadvantages:
    - Creating detailed time diagrams for highly complex systems can be time-consuming.
    - Time diagrams may not capture all nuances of system behavior.

Outcomes of the Topic Taught:

- Learning about time diagrams equips individuals with a powerful tool to visualize, analyze, and manage the temporal aspects of systems and processes. This can lead to more efficient and reliablesystem designs.

Potential Applications & Implementation in Industries:

- Time diagrams find applications in various industries, including electronics, telecommunications, software development, and manufacturing. They are used to analyze and optimize timing relationshipsin systems, ensuring correct operation and adherence to specified time constraints. For example, in telecommunications, time diagrams are used to ensure the synchronization of signals and data packets.

Interaction Diagram

Introduction of the Topic:

Interaction diagrams are a crucial aspect of the Unified Modeling Language (UML) used in software engineering and system design. They provide a visual representation of how objects or components in asystem interact and communicate with each other during the execution of a use case or a scenario.

What is the Need of Studying the Topic:

Studying interaction diagrams is essential because they help in visualizing and documenting the dynamic behavior of a system. They are valuable tools for understanding and improving the communication and collaboration among stakeholders in software development projects.

Detailed Discussion of the Topic:

- **Interaction Diagrams:**
    - There are two main types: Sequence Diagrams and Communication Diagrams.
    - Sequence Diagrams:
        - Show the sequence of interactions between objects or components over time.
        - Use vertical lifelines to represent objects, and arrows to represent messages ormethod calls.
    - Communication Diagrams:
        - Emphasize the relationships and connections between objects.
        - Use lines to connect objects and show the flow of messages.

Interaction Diagrams

Sequence diagrams and collaboration diagrams• both of which are called interaction diagrams• are two of the five diagrams used in the UML for modeling the dynamic aspects of systems. An interaction diagram shows an interaction, consisting of a set of objects and their relationships, including the messages that may be dispatched among them. A sequence diagram is an interaction diagram that emphasizes the time ordering of messages; a collaboration diagram is an interaction diagram that emphasizes the structural organization of the objects that send and receive messages.

Terms and Concepts

An *interaction diagram* shows an interaction, consisting of a set of objects and their relationships, including the messages that may be dispatched among them. A *sequence diagram* is an interaction diagram that emphasizes the time ordering of messages. Graphically, a sequence diagram is a table that shows objects arranged along the X axis and messages, ordered in increasing time, along the Y axis. A *collaboration diagram* is an interaction diagram that emphasizes the structural organization of the objects that send and receive messages. Graphically, a collaboration diagram is a collection of vertices and arcs.

Contents

Interaction diagrams commonly contain

· Objects

· Links

· Messages

A. Sequence Diagrams

A sequence diagram emphasizes the time ordering of messages. As Figure 18-2 shows, you form a sequence diagram by first placing the objects that participate in the interaction at the top of your diagram, across the X axis. Typically, you place the object that initiates the interaction at the left, and increasingly more subordinate objects to the right. Next, you place the messages that these objects send and receive along the Y axis, in order of increasing time from top to bottom. This gives the reader a clear visual cue to the flow of control over time.

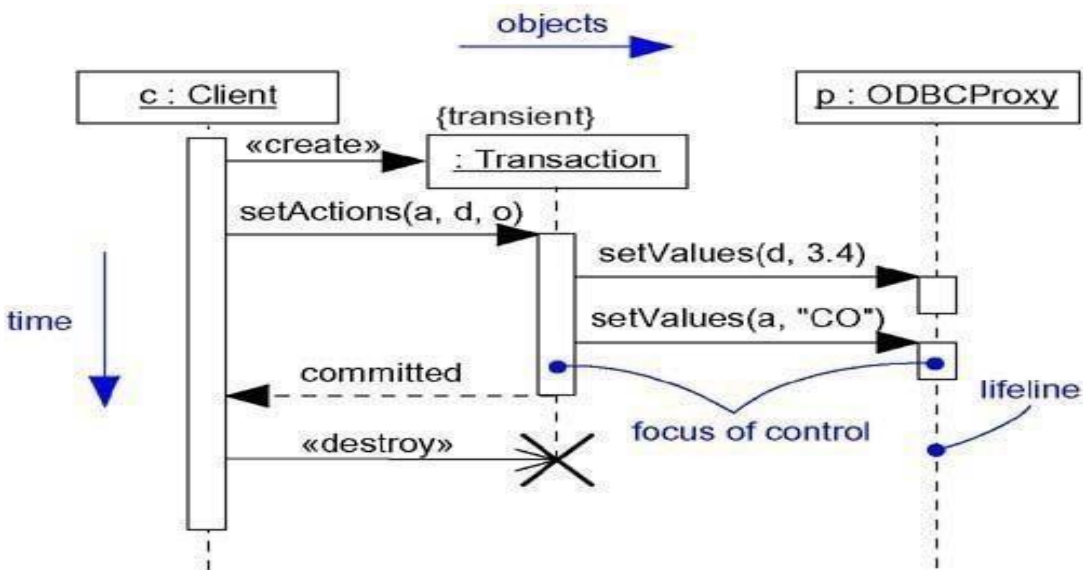


Figure 18-2 Sequence Diagram

Sequence diagrams have two features that distinguish them from collaboration diagrams.

You can specify the vitality of an object or a link by marking it with a new destroyed, or transient constraint

First, there is the object lifeline. An object lifeline is the vertical dashed line that represents the existence of an object over a period of time.

Second, there is the focus of control. The focus of control is a tall, thin rectangle that shows the period of time during which an object is performing an action, either directly or through a subordinate procedure

Flowchart/Algorithm/Models:

- Interaction diagrams are graphical models that do not involve specific flowcharts or algorithms. Theyare used to represent dynamic behaviour.

Characteristics Related to the Topic:

- Interaction diagrams focus on how objects or components collaborate to achieve specific functionality.
- They depict the order and timing of messages exchanged during the execution of a use case or scenario.

Advantages & Disadvantages:

- Advantages:
    - Improve the understanding of the dynamic behavior of a system.
    - Enhance communication among development team members and stakeholders.
    - Aid in the identification of potential design flaws or communication issues.
- Disadvantages:
    - Can become complex and overwhelming for large and intricate systems.
    - The level of detail can vary, making it important to strike a balance betweencomprehensiveness and simplicity.

Outcomes of the Topic Taught:

- Learning about interaction diagrams equips individuals with a powerful tool to model, understand, and communicate the dynamic behavior of a system. This leads to improved system design and more efficient development.

Potential Applications & Implementation in Industries:

- Interaction diagrams are widely used in the software industry for system design and documentation. They help in understanding and improving the behavior of software systems, and are especially valuable in industries like software development, telecommunications, and financial services, where the behavior of complex systems is critical to success.

Package Diagram

Introduction of the Topic:

- A package diagram is a diagram in the Unified Modeling Language (UML) that helps organize and visualize the structure and dependencies of a system by grouping related elements into packages. It's an important tool for designing, documenting, and managing the architecture of complex software systems.

What is the Need of Studying the Topic:

- Studying package diagrams is essential for effectively organizing and managing the complexity of largesoftware systems. They provide a structured way to represent the components, modules, and relationships within a system.

Detailed Discussion of the Topic:

- Package Diagram:
    - Organizes elements into packages, which are represented as rectangles.
    - Shows dependencies between packages using arrows or lines.
    - Can contain other UML diagrams, classes, interfaces, and other elements.

Flowchart/Algorithm/Models:

- Package diagrams are a visual representation and do not involve specific flowcharts or algorithms.They are used to model and manage the architectural structure of a system.

Characteristics Related to the Topic:

- Package diagrams emphasize the organization of elements within a system, allowing for a high-levelview of system architecture.

- Dependencies between packages are shown to illustrate the relationships between different parts of thesystem.

Advantages & Disadvantages:

- Advantages:
  - Improve system organization and maintainability by providing a clear structure.
  - Enhance communication and collaboration among development team members.
  - Facilitate modularization and component-based development.
- Disadvantages:
  - May not capture every detail of system behavior.
  - The level of abstraction in package diagrams should be carefully chosen to avoidovercomplexity.

Outcomes of the Topic Taught:

- Learning about package diagrams equips individuals with a powerful tool to manage the architecture ofa software system, improving its organization and maintainability. This leads to more efficient system design and development.

Potential Applications & Implementation in Industries:

Package diagrams are widely used in industries such as software development, systems engineering, and IT management to visualize, design, and maintain the architecture of software systems. They help in organizing,communicating, and managing the structure of complex systems, ensuring scalability and maintainability.

QUESTION BANK

Two Marks Questions

- What is the primary purpose of a use case in software development?
- How do use cases help in improving communication among stakeholders?

Use Case Diagrams:

- What are actors in a use case diagram, and how are they represented?
- Explain the role of the system boundary in a use case diagram.

Activity Diagrams:

- How are actions represented in an activity diagram?
- What is the main objective of using an activity diagram in software modeling?

State Machine:

- Describe the basic components of a state machine.
- How does a state machine differ from an activity diagram in UML?

Process and Thread:

- What is the fundamental difference between a process and a thread in the context of operating systems?
- How does multi-threading improve the performance of software applications?

Event and Signals:

- Explain the concept of an event in software design.
- How are signals used in real-time and embedded systems?

Time Diagram:

- What is the main purpose of creating a time diagram?
- How can time diagrams help in identifying timing issues in a system?

Interaction Diagram:

- Differentiate between Sequence Diagrams and Communication Diagrams.

- How do interaction diagrams facilitate collaboration among development team members?

Package Diagram:

- What is the primary role of a package in software architecture?
- How can package diagrams enhance the maintainability of a software system?

FIVE MARK QUESTIONS

Use Cases and Use Case Diagrams:

- Explain the purpose of a use case diagram in software development. Describe the key components of a use case diagram and provide an example of a real-world scenario represented using use cases and actors.

Activity Diagrams:

- Create an activity diagram for a simple online purchase process, including actions, decisionpoints, and control flow. Explain how activity diagrams can help in modeling complex workflows and improving system design.

State Machine and State Transition Diagrams:

- Illustrate a state transition diagram for a traffic light system with various states and transitions.Explain the concept of hierarchical states and how they can be used to model complex behaviors in a state machine.

Process and Thread:

- Compare and contrast processes and threads in operating systems. Discuss the advantages anddisadvantages of multi-threading and provide an example of a situation where using threads would be beneficial.

Event and Signals:

- Define what events and signals are and explain how they are used in event-driven programming. Provide a real-world example of a system that relies heavily on event handlingand describe its key components.

Time Diagram:

- Create a time diagram representing the scheduling of tasks in a multi-core processor system. Discuss the importance of time diagrams in understanding the timing constraints of concurrentsystems and real-time processes.

Interaction Diagrams:

- Construct a sequence diagram depicting the interaction between various objects in an online shopping system when a user places an order. Explain the purpose of sequence diagrams andtheir role in system design and communication.

Package Diagram:

- Design a package diagram for a software application that includes different software modules and libraries. Discuss the advantages of using package diagrams for managing the architectureof a complex software system and how they aid in system scalability and maintainability.

7/10 Mark Questions

Use Cases and Use Case Diagrams (7 marks):

- Design a use case diagram for a library management system. Include actors, use cases, and their relationships. Explain how this diagram helps in requirements analysis and system design. Discuss the role of extending and including use cases in the context of your diagram.

Activity Diagrams (7 marks):

- Create an activity diagram for the process of booking a flight ticket, covering various scenarios such as seat selection, payment processing, and confirmation. Explain the use of forks, joins, and decision nodes in your diagram and how they affect the flow of activities.

State Machine (7 marks):

- Develop a state machine diagram for a vending machine. Define states, transitions, and events,and show how the machine responds to user inputs and internal states. Explain the concept of a superstate and how it simplifies the representation of complex behaviors.

10-Mark Questions:

Process and Thread (10 marks):

- Compare and contrast processes and threads in operating systems. Discuss their individual advantages and disadvantages in detail. Provide a case study where the choice between processes and threads has a significant impact on system performance and reliability. Justifyyour decision.

Event and Signals (10 marks):

- Explore the role of events and signals in event-driven programming and real-time systems. Describe in detail how event handling and signal processing work. Provide examples of bothsynchronous and asynchronous events, and explain how they influence the control flow of a program.

Time Diagram (10 marks):

- Design a time diagram to illustrate the scheduling and execution of multiple tasks on a real-time embedded system. Include timing constraints, dependencies, and critical paths. Discussthe challenges of ensuring real-time compliance and meeting deadlines in such systems.

Interaction Diagrams (10 marks):

- Create a sequence diagram and a communication diagram to depict the interaction between objects in a banking system during a money transfer operation. Compare and contrast the twotypes of interaction diagrams, emphasizing their strengths and weaknesses in representing complex system interactions. Discuss how these diagrams facilitate collaboration among development team members.

Package Diagram (10 marks):

- Develop a package diagram for a complex e-commerce platform, showcasing the different layers, modules, and components involved. Explain how packages help in managing systemcomplexity and improving maintainability. Discuss the strategies for packaging and dependency management in large-scale software projects.

CONCEPTUAL QUESTIONS

Use Cases and Use Case Diagrams:

- What is the fundamental purpose of a use case in software engineering, and how does it differfrom a use case diagram?
- How do use case diagrams help in representing the functional requirements of a software system and in improving communication among project stakeholders?

Activity Diagrams:

- Explain the primary objectives of an activity diagram and how it helps in modeling complexworkflows in a system.
- How does the concept of concurrency and parallelism manifest in activity diagrams, and whyis it important in system modeling?

State Machine:

- Define the core components of a state machine and their roles in modeling the behavior of systems.
- Discuss the significance of hierarchical states and transitions in state machine diagrams, andprovide an example where hierarchical states are beneficial.

Process and Thread:

- What are the fundamental differences between a process and a thread in the context of operating systems, and how do they relate to multitasking and parallelism?
- Explain how thread synchronization and communication mechanisms, such as semaphores andmutexes, enable coordinated execution in concurrent systems.

Event and Signals:

- Define what events and signals are and describe their roles in event-driven programming andreal-time systems.
- Discuss how event-driven programming fosters responsiveness and flexibility in software applications and how it differs from traditional sequential programming.

Time Diagram:

- What is the primary purpose of a time diagram, and how does it help in understanding timingconstraints and the behavior of systems over time?
- Explain the relevance of time diagrams in the context of real-time and embedded systems andhow they aid in ensuring system predictability.

Interaction Diagrams:

- Compare and contrast Sequence Diagrams and Communication Diagrams, highlighting theirprimary purposes and representations.
- How do interaction diagrams improve communication and collaboration among developmentteam members and stakeholders in software projects?

Package Diagram:

- Define what packages are in software design and architecture and explain how they help in managing the complexity of large systems.
- Discuss the best practices for organizing packages, including strategies for minimizing dependencies and ensuring modularity in software systems.

<u>INDUSTRY BASED QUESTIONS</u>

Use Cases and Use Case Diagrams:

Q1: In the healthcare industry, how can use case diagrams help in improving patient care and hospitalmanagement?

Answer: Use case diagrams in healthcare can model scenarios such as patient admission, discharge, or medication administration. They help in streamlining processes, reducing errors, and enhancing patient care. Forexample, a use case diagram can represent the process of patient check-in, ensuring a systematic approach and minimizing wait times.

Q2: Explain how use case diagrams are used in the banking sector to enhance customer experience andoperational efficiency.

Answer: Use case diagrams in banking can represent actions like ATM transactions, account inquiries, and fund transfers. By visualizing these interactions, banks can design user-friendly interfaces and optimize their systems. This improves customer experience and reduces operational errors, benefiting both customers and the bank.

Activity Diagrams:

Q3: In the manufacturing industry, provide an example of how activity diagrams can be used to model andoptimize production processes.

Answer: Activity diagrams can represent the steps involved in manufacturing a product. For instance, inautomobile production, an activity diagram can illustrate the sequence of actions from assembly to quality control, helping identify bottlenecks and improving overall efficiency.

Q4: How do activity diagrams support project management in the construction industry?

Answer: Activity diagrams can represent construction project workflows, showing tasks, dependencies, andresource allocation. Project managers can use these diagrams to schedule activities, allocate resources efficiently, and ensure the project stays on track.

State Machine:

Q5: In the aviation industry, how are state machines applied to ensure the safety of flight control systems?

Answer: State machines model the behavior of flight control systems, helping ensure safe operations. For example, a state machine can depict the various states an aircraft's landing gear can be in, facilitating precisecontrol during takeoff, landing, and maintenance.

Q6: Discuss the role of state machines in the gaming industry, specifically in character behavior and game AI.

Answer: State machines are used to model character behaviors in video games. Different states represent actions like walking, running, or attacking. Transitions between states are based on game events or player input,allowing characters to respond realistically to in-game situations.

Process and Thread:

Q7: How do processes and threads play a critical role in the software development industry when designingmulti-platform applications?

Answer: Processes and threads are essential for developing multi-platform software. Processes can represent platform-specific components, while threads handle parallelism within components. This architecture ensures compatibility across diverse platforms, enhancing software reach.

Q8: Explain the significance of thread pools in optimizing web server performance in the IT industry.

Answer: Thread pools help web servers efficiently manage client requests. Instead of creating a new thread for each request, a pool of threads is maintained, reducing the overhead of thread creation and ensuring optimal resource utilization.

Event and Signals:

Q9: In the telecommunications industry, how do events and signals facilitate real-time network management and fault detection?

Answer: Events and signals are used to notify network administrators of faults, performance issues, or security breaches in real-time. For example, a signal can trigger an alert when a network node exceeds a predefined threshold, allowing immediate action.

Q10: How are events and signals utilized in the entertainment industry to create immersive experiences in theme park attractions?

Answer: Events and signals trigger actions in theme park attractions, synchronizing audio, lighting, and mechanical effects. For instance, a sensor detecting a ride vehicle's position can signal the audio system to playa corresponding sound effect, enhancing the overall experience.

Time Diagram:

Q11: In the finance sector, how do time diagrams aid in tracking and optimizing trading activities in high-frequency trading systems?

Answer: Time diagrams track the timing of trade executions, helping analyze trading strategies and latency issues. They provide insights into order execution and latency, crucial for optimizing high-frequency trading systems.

Q12: Explain how time diagrams are used in the transportation and logistics industry to schedule and track the movement of goods.

Answer: Time diagrams assist in visualizing the transportation and delivery schedule of goods. They show the pickup, transit, and delivery times for shipments, enabling logistics providers to optimize routes and meet delivery deadlines efficiently.

Interaction Diagram:

Q13: How do interaction diagrams help improve customer service in the telecommunications industry, particularly when troubleshooting network issues?

Answer: Interaction diagrams model the communication between customer service agents and customers when diagnosing network problems. They ensure a systematic and efficient troubleshooting process, leading to faster issue resolution and improved customer satisfaction.

Q14: In the e-commerce industry, explain how interaction diagrams support the implementation of onlineshopping carts and secure payment processes.

Answer: Interaction diagrams depict the flow of actions between customers, shopping carts, and payment gateways. They help ensure a seamless online shopping experience, guiding customers through the selection ofproducts, addition to the cart, and secure payment processing.

Package Diagram:

Q15: How do package diagrams enhance the management of software components and libraries in the softwaredevelopment industry?

Answer: Package diagrams organize software components into logical units, making it easier to manage dependencies, version control, and modularity. This ensures a more efficient and maintainable software development process.

Q16: Explain how package diagrams are used in the healthcare IT sector to structure and manage electronichealth record (EHR) systems.

Answer: Package diagrams in healthcare IT represent modules like patient records, prescriptions, and billing.They help in organizing EHR components, enabling interoperability, data security, and adherence to industrystandards.

Questions asked in competitive examinations with answers.

Use Cases and Use Case Diagrams:

Q1: What is a use case in software engineering, and why is it important?

- Answer: A use case represents a specific interaction or functionality that a system provides to an actor.It helps in defining the system's behavior from the user's perspective and serves as a basis for system design and testing.

Q2: In a use case diagram, what is the role of an actor? Provide an example of an actor in a use case diagram.

- Answer: Actors represent entities external to the system that interact with it. An example of an actor in an airline reservation system is "Passenger."

Activity Diagrams:

Q3: How are actions and control flow represented in an activity diagram?

- Answer: Actions are represented as rectangles with rounded corners, and control flow is depicted using arrows to show the sequence and dependencies between actions.

Q4: Explain the purpose of swimlanes in activity diagrams.

- Answer: Swimlanes are used to represent different organizational units or entities responsible for specific actions in a process. They help in visualizing the roles and responsibilities in a workflow.

State Machine:

Q5: What is the primary purpose of a state machine diagram, and how does it differ from a flowchart?

- Answer: A state machine diagram models the behavior of an object or system by depicting its states,transitions, and events. It focuses on state changes over time, whereas a flowchart represents a sequence of steps.

Q6: How are hierarchical states used in a state machine diagram, and what advantages do they offer?

- Answer: Hierarchical states allow complex behavior to be broken down into smaller, manageable parts.This promotes modularity and simplifies the representation of complex systems.

Process and Thread:

Q7: Explain the concept of a thread in the context of multitasking. How does it differ from a process?

- Answer: A thread is the smallest unit of execution within a process, while a process is an independent program with its own memory space and resources. Threads within the same process share memory, making them more lightweight than processes.

Q8: What is thread synchronization, and why is it important in concurrent programming?

- Answer: Thread synchronization ensures that multiple threads can coordinate their actions to prevent data races and inconsistencies when accessing shared resources. It's vital for maintaining data integrityin concurrent systems.

Event and Signals:

Q9: What is an event in software design, and how does it affect program flow?

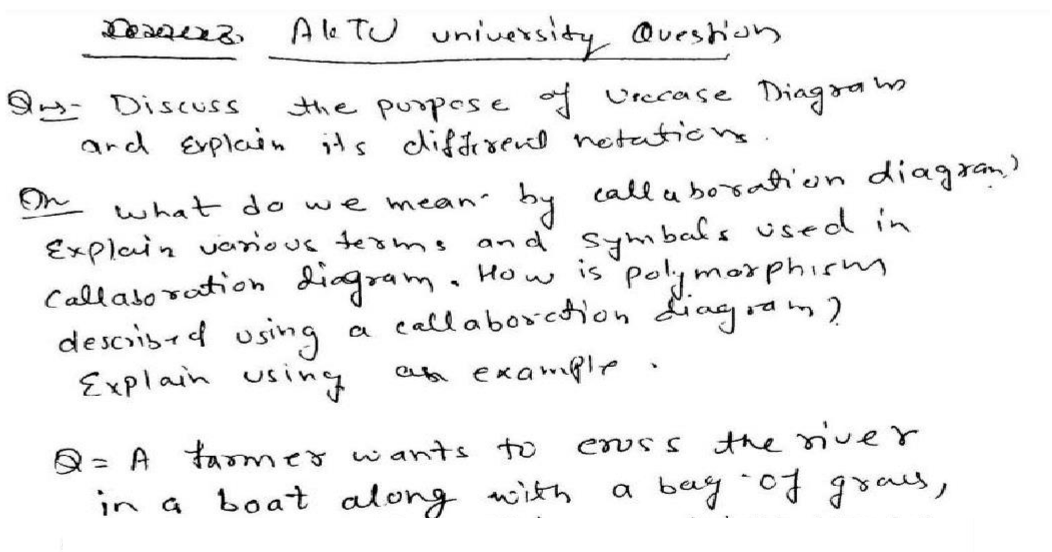
- Answer: An event is an occurrence that triggers a response in a program. It can interrupt the normalflow of execution, leading to event-driven behavior.

Q10: In the context of real-time systems, how are signals used for inter-process communication and synchronization?

- Answer: Signals are used to notify processes about specific events or conditions. They enable inter-process communication and allow processes to respond to events in real time.

These sample questions cover key concepts from the topics you mentioned and can serve as a foundation for competitive examination preparation.

UNIVERSITY QUESTIONS

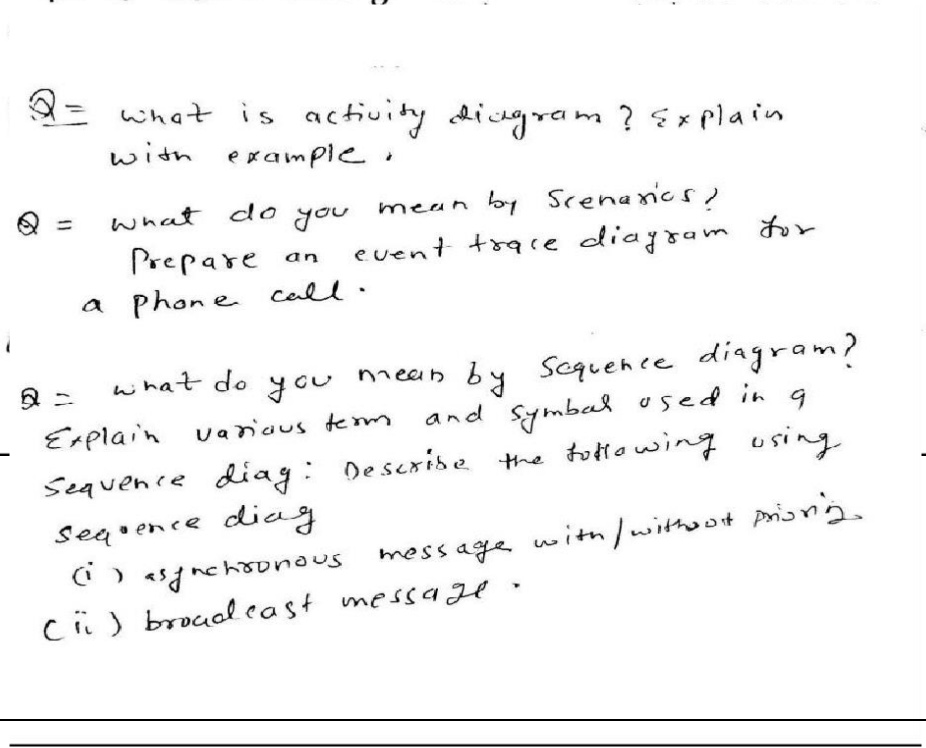


[illegible] AKTU university Question

Q- Discuss the purpose of Usecase Diagrams and Explain its different notations.

Qn what do we mean by callaboration diagram? Explain various terms and symbols used in callaboration diagram. How is polymorphism described using a callaboration diagram? Explain using an example.

Q = A farmer wants to cross the river in a boat along with a bag of grass,



Q = what is activity diagram? Explain with example.

Q = what do you mean by Scenarios? Prepare an event trace diagram for a phone call.

Q = what do you mean by Sequence diagram? Explain various term and symbol used in a Sequence diag: Describe the following using Sequence diag

(i) asynchronous message with/without priority.

(ii) broadcast message.

Q. No.	Question	Marks	CO
University Related Questions			
1	What is Procedure Oriented Approach?	2	CO1
2	Write short notes on OOM and OOD?	2	CO1
3	What is Encapsulation?	2	CO1
4	What is Generosity?	2	CO1
5	What is Interface?	2	CO1
6	Differentiate between dependencies and associations?	2	CO1
7	What is Process view?	2	CO1
8	What is Design view?	2	CO1
9	What is object and the notation of object's?	2	CO1
10	What is Object, Class and Instance?	2	CO1
11	What are the main differences between procedure-oriented languages and object-oriented languages?	**5**	**CO1**
12	What are the basic OOPs principles?	**5**	**CO1**
13	Difference between Top down and bottom up approaches for a given project?	**5**	**CO1**
14	What do you meant by static and dynamic modeling?	**5**	**CO1**
15	Why do we need OOPs?	**5**	**CO1**
16	What is abstraction?	**5**	**CO1**
17	Using diagrams, explain the following concepts as applied in Object Oriented Analysis and Design. Object, class, polymorphism, encapsulation and	**5**	**CO1**
18	Explain about object interaction and collaboration?	**5**	**CO1**
19	What is meant by object? How are these created?	**5**	**CO1**
20	What is inheritance? Discuss types of inheritance with example.	**5**	**CO1**
21	What is the difference between Multilevel and Muliple Inheritance?	**5**	**CO1**
22	List the advantage offered by object-oriented systems development.	**5**	**CO1**
23	What is an abstract class? Is it possible that an abstract class is inherited by another class?	**5**	**CO1**
24	When do we use the protected visibility specifier to a class member?	**5**	**CO1**
25	What is information hiding?	**5**	**CO1**
26	What do you understand by Object identity? Explain with an example.	**5**	**CO1**

27	What do you mean by encapsulation ? How does the object-oriented concept of message passing help to encapsulate the implementation of an object,including its data?	**5**	**CO1**
28	Explain all basic concepts of object oriented programming.	**5**	**CO1**
29	What are the principles of modeling ? What is the importance of modeling ?	**5**	**CO1**
30	What are the 3 primary elements of collaboration diagram?	**5**	**CO2**
31	What is the purpose of a collaboration diagram? What are the symbols used in collaboration diagram?	**5**	**CO2**
32	Define the Similarities and difference between Sequence diagram and Collaboration diagram	**5**	**CO2**
33	What is UML ? List all building blocks of UML. Explain all types of things used in UML.	**10**	**CO1**
34	Define an abstract class. When is it required to create an abstract class ? Explain it with an example.	**10**	**CO1**
35	Define link and association. Discuss the role of link and association in object modeling with suitable example.	**10**	**CO1**
36	Explain generalization, aggregation and association in detail.	**10**	**CO1**
37	Give the conceptual model of UML. Use some example to illustrate.	**10**	**CO1**
38	Why UML required ? What are the basic architecture of UML ?	**10**	**CO1**
39	What do you mean by a collaboration diagram ? Explain various terms and symbols used in a collaboration diagram. How polymorphism is described using a collaboration diagram ? Explain using an example.	**10**	**CO2**
40	What do you mean by polymorphism ? Is this concept only applicable to object-oriented systems ? Explain.	**10**	**CO2**
41	What are the three ways to apply UML?	**10**	**CO2**
42	State which UML diagrams give a static view and which give a dynamic view of a system.	**10**	**CO2**
43	Describe the UML and need of UML and explain the benefits of UML?	**10**	**CO2**
44	Define the building blocks of UML	**10**	**CO2**
45	Explain behavioral Diagrams and any three types of behavioral Diagrams?	**10**	**CO2**
46	Explain structural diagrams and any three types of structural diagrams?	**10**	**CO1**
47	What is modeling? What are the advantages of creating a model?What are the different views that are considered when building an object-oriented software system?	**10**	**CO1**
48	What are Relationships? Explain a different kind of Relationships?	**10**	**CO2**

49	Enumerate the graphical notations used in structural things are most widely used in UML.	**10**	**CO2**
50	What are the four basic relationships defined in UML? Give suitable examples for usage of each type of relationships?	**10**	**CO2**
51	What is class diagram in UML with example?Define the purpose of class diagram.	**10**	**CO2**
52	Describe the notation used for the collaboration diagram.	**10**	**CO2**
53	What are the different types of Messages in Sequence Diagrams,Draw Sequence diagram of ATM.	**10**	**CO2**