Chapter 1

Introduction

1.0 OBJECTIVES

After studying this chapter you will be able -

- To understand the need of models.
- To know what is UML.
- To know the features of UML.
- To understand the need for UML.
- To know what UML is not.

1.1 INTRODUCTION

Models are representations of reality which help simplify the reality. It is not different in the software development field. The software under development can be complex. So the models of software can be used. One such modeling technique is the Unified Modeling Language.

1.2 WHY MODELS?

Models are created to represent reality in a simplified way. They provide a structure for problem solving. They help in understanding how one can proceed with the problem at hand. They also allow to experiment with multiple solutions. Since the models are created before the actual development of the system, we can understand different possibilities, problems, options etc. This also leads to decrease in the development costs. Since the time will not be wasted in trials and errors the product will be ready in lesser time. Models also help manage the complexity of the problem so the planning of the development, allocation of the resources like machines, programmers, testers can be done easily.

When models are created, a number of views of the system are described. The different views give different perspectives of the system.

For example,

The car dashboard and its features interest the car driver. He needs to know which button to press to switch on the inner lights and so on. But the electronics person is not concerned about the dashboard in a similar way. He is interested in the circuitry behind the dashboard. He wants to know which buttons to connect to which instruments, gadgets so that when the button is pressed on the dashboard the driver gets the expected operation. The design person is interested in the look and feel of the dashboard but not all concerned with what goes on behind the panel. Thus everybody looks at the same object but from different perspectives.

Each view has information that is unique to that view. Each view has the information that appears in other views. The information that appears in more than one view is consistent. This information has to be consistent because we are creating views of the same system. This leads to the need for a variety of views depicted in a variety of diagrams.

1.3 WHAT IS UML?

The Unified Modeling language, popularly known as the UML is a visual modeling language that helps system developers to create blueprints that captures their vision in a standard, easy to understand form, and gives a mechanism to effectively share and communicate their visions with others.

The Object Management Group gives the formal definition of the UML as

The UML is a graphical language for specifying, visualizing, constructing and documenting the artifacts of the software system.

Specifying – UML provides the means to model precisely, unambiguously and completely the system in question.

Visualizing – UML provides graphical notation which articulates and ambiguously communicates the overall view of the system.

Constructing - UML provides the 'design' dimension to the models built. These are language independent and can be implemented in any programming language.

Documenting – every software project involves a lot of documentation from the inception phase to the deliverables.

OMG WE SET THE STANDARD UNIFIED MODELING LANGUAGE ™ UML®

1.2-1.3 Check your progress

Fill in the blanks

1. The different _____ give different ………………… of the system.
2. The information that appears in more than one view is ……………
3. The UML is a ………………. language for specifying, visualizing, constructing and documenting the artifacts of the software system.

1.4 FEATURES OF UML

The UML has the following features:

- **Syntax only –** the UML is a language. It tells what model elements are available with different diagrams and how to use them.
- **Comprehensive –** it can be used to model anything even though the modeling syntax aimed primarily at creating models of software-based systems.
- **13 diagrams –** there are 13 different diagrams provided which help model different perspectives of the system. It is not necessary to use all of them in practice but they are there to be used.
- **Language-independent –** it has no bindings to any high-level language. The tool which will be used can do the necessary bindings with the chosen language. The same set of models can be used to develop systems in several languages.

- **Process independent –** it has no binding with the process by which the models are created. Any process can be used for creating models.
- **Tool-independent –** there are restrictions on the vendors who provide the tools for drawing UML diagrams. Each vendor can do value addition to the visual modeling of the diagrams.
- **Well documented –** the documentation for UML diagrams is available. The UML notation guide is readily available which gives the details of the notation used as well as example using the notation.

1.5 NEED FOR UML

The UML is a modeling language and not a notation system. Since it is a language it has predefined structure as well as a repository for the different notations used in it. In English language, when a sentence is written, it has a fixed structure. "I have a book" cannot be written as "A book have I" or "Have a book I".

1.6 WHAT UML IS NOT

- UML is not a notation, but it is a language.
- UML is not owned by anyone. It is open to be used by anyone who wishes to use it. It is not proprietary.
- UML is not process or a method. UML encourages the use of object-oriented techniques and iterative software development lifecycles.
- UML is not difficult. It is large, but one need not know it entirely. Also there is no need to use or understand every small thing in it.
- UML is not time consuming. If properly used, use of UML cuts down the development costs. At the same time it gives the advantage of easy understanding and communication, increased productivity and better quality.
- UML is not limited. It is flexible enough to allow newer vocabulary (concepts, words and terms), properties (additional information about the words) and semantics (language rules) which are required for a specific domain.

1.4-1.6 Check your progress

True or False

1. The UML is language independent.
2. The UML does not have any predefined structure.
3. The UML is owned by a particular organization.
4. The UML is not limited

1.7 SUMMARY

In this chapter, initially the need for models is explained. There are a variety of modeling techniques available. The UML is one such technique which plays an important role in the object oriented software development.

1.8 CHECK YOUR PROGRESS – *ANSWERS*

1.2-1.3

1. views, perspectives
2. consistent
3. graphical

1.4 -1.6

1. True
2. False
3. False
4. True

1.9 QUESTIONS FOR SELF - STUDY

1. Why are models needed?
2. Why are different views considered?
3. What is UML?
4. What are the features of UML?

1.10 SUGGESTED READINGS

1. *Unified Modeling Language User Guide by* Grady Booch,James Rumbaugh, Ivar Jacobson
2. *UML 2 For Dummies* by Michael Jesse Chonoles, James A. Schardt

***

NOTES

NOTES

Chapter 2

Review of Object Orientation

2.0 OBJECTIVES

To revise the following object oriented concepts -

- Objects
- Classes
- Encapsulation
- Abstraction
- Inheritance
- Polymorphism
- Message passing

2.1 INTRODUCTION

The UML can be used in a better way with the object oriented technologies. So there is a need to revise the object concepts. All the important concepts will be covered in this chapter.

2.2 OBJECT

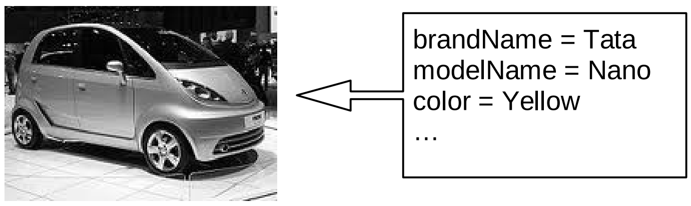
Every day we come across many things. The things can be a tangible thing like a car or a printer, a role like an employee or a boss, an incident like a flight or an overflow, interaction like contract or a sale or a specification like a color or a shape. Each of these things has some kind of identity, state and behavior.

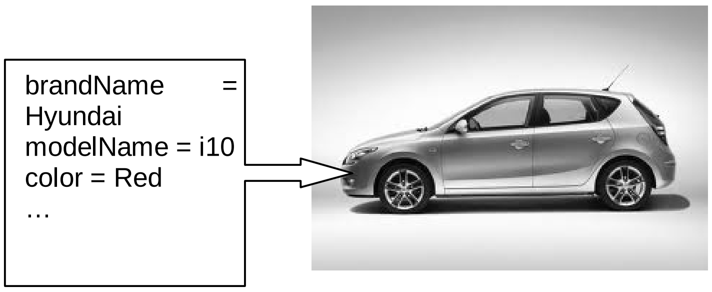
The identity is the property of an object that distinguishes it from other objects like myCar, yourCar.

The state describes the data stored in that object like for a car we can specify the brand name as Tata, the model name as Nano, the color as yellow for myCar and the brand name Hyundai, the model name as i10, the color as red for yourCar and so on. The state or the structure is described by the properties or the attributes.

The behavior describes the operations the object carries out like start a car, stop a car, change gear, accelerate, and apply brake. The behavior is described by the operations or the methods.

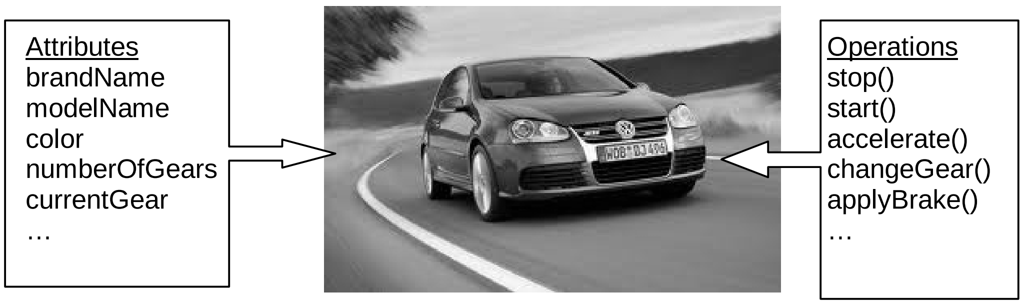
All the cars have a brand name, a model name, a color and so on. All the cars can be started, stopped, accelerated. All the features that describe the state and the behavior of cars are similar. So we need not define them separately.





2.3 CLASS

There can be a large number of cars. We need not define each of them separately. Instead we can describe them together by grouping all the cars under a pattern, template or blueprint. This pattern is known as a class. It creates a group of structurally identical items. e.g. car. A car can have attributes like a brand name, a model name, a color. A car can have operations like to stop, to start, to accelerate, to change gear. A class consists of both a pattern and a mechanism for creating items based on that pattern. The items created or manufactured using the class are called instances. e.g. myCar, yourCar.



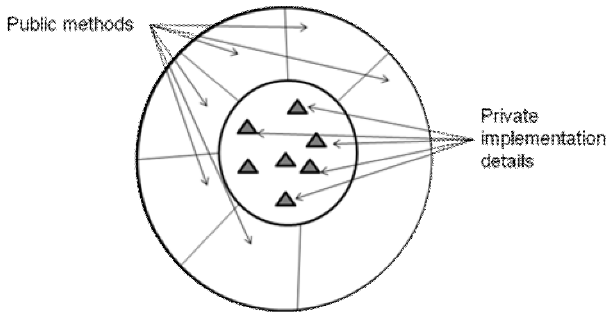
2.2-2.3 Check your progress

Fill in the blanks

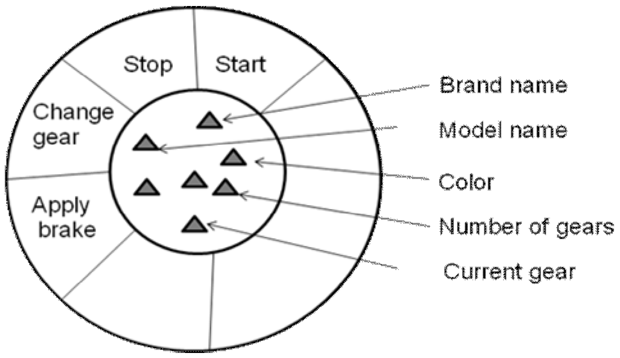
1. The ………………….. is the property of an object that distinguishes it from other object
2. The behavior describes the …………….. the object carries out.
3. A class creates a group of ……………… items.
4. The items created or manufactured using the class are called ………………………..

2.4 ENCAPSULATION

The wrapping of data and functions into a single unit called a class is known as encapsulation. The data is not directly accessible to the outside world. Only the operations or functions or methods wrapped in the class can access it. This gives protection and security to the data. The data is insulated from direct access. This achieves a way to hide information or data.



For the car class the following diagram shows the encapsulation where the operations are around the attributes. The attributes are not directly visible or open to the outside. The operations above them form an insulating cover or wrapper around them.



2.5 ABSTRACTION

Abstraction is the act of representing the essential features without including the background details or explanations. This is a fundamental principle of modeling. A system model is created at different levels, starting at the higher level and adding more levels are more and more details become known about the system. A model can be viewed at several levels, each giving different details about the system. So abstraction is looking only at the information that is relevant at the time by hiding details so that there is no confusion in understanding the system.

For example,

A car is an abstraction which represents the features i.e. the attributes like brandName and modelName and the operations like start and change gear. Also a car represents all possible models of all possible cars.

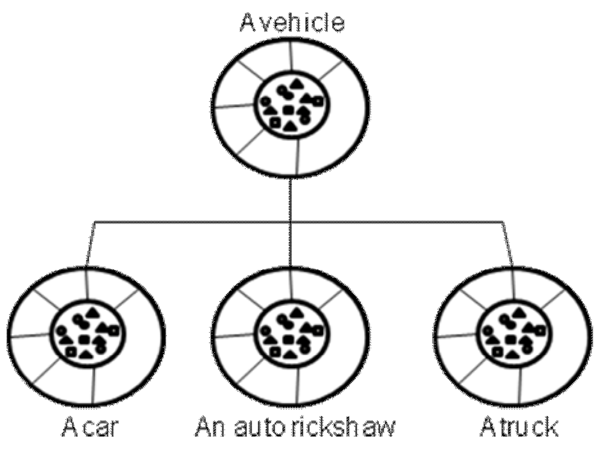
2.6 INHERITANCE

The process by which objects of one class acquire the properties of objects of another class is known as inheritance. This supports hierarchical classification. This provides reusability. It makes it possible to add additional features to an existing class without modifying it. This helps avoid duplication.

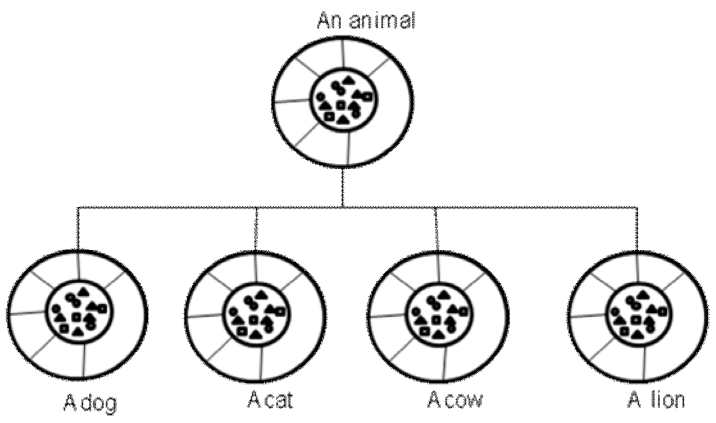
A class known as the subclass or the child class can inherit attributes and operations from another class known as the superclass or the parent class. The subclass provides the specialized behavior whereas the superclass provides the generalized behavior.

For example:

A car is a vehicle. An auto rickshaw is a vehicle. A truck is a vehicle. The superclass is vehicle. The subclasses are car, auto rickshaw, truck.



A dog is an animal. A cat is an animal. A cow is an animal. A lion is an animal. The superclass is animal. The subclasses are dog, cat, cow, and lion.



2.4-2.6 Check your progress

True or False

1. The data is directly accessible to the outside world due to encapsulation.
2. Abstraction is looking only at the information that is relevant at the time by hiding details.
3. Encapsulation supports abstraction.
4. Inheritance supports hierarchical classification and provides reusability.

2.7 POLYMORPHISM

Polymorphism means the ability to take more than one form. The polymorphism is of two types:

2.7.1 Function polymorphism

An operation may exhibit different behavior in different instances depending upon the data used in the operation. Here a single name is used to perform different tasks. The same method used in a superclass can be overridden in subclasses to give a different functionality.

For example,

All animals make some kind of a sound. But the sounds of different animals are termed differently. A cat mews. A dog barks. A lion roars.

All polygons occupy area. A triangle, a rectangle, a square are all polygons. The superclass polygon has a method getArea(). For getArea() in subclass triangle the calculation is done by the formula ½ of the product of the base and height. For getArea() in subclass rectangle the calculation is done by the formula product of the length and breadth.

2.7.2 Operator polymorphism

This is the process of making an operator to exhibit different behaviors in different instances.

For example,

'+' operator when used with integers, it gives the result as the addition of the two integers. We might wish '+' to add two matrices or two vectors.

2.8 MESSAGE PASSING

A message is a request for execution of a procedure that invokes a function in the receiving object that generates the desired result.

Message passing achieves communication between a set of objects with each other. This communication is established by sending and receiving information. This can be achieved by specifying the name of the object, the name of the function (message) and the information to be sent.

For example,

A television may be operated by a remote. When the ON button on the remote is pressed; the television should get switched on. Similarly when the OFF button on the remote is pressed; the television should get switched off. The volume increase/decrease button on the remote is pressed; similar action should be activated by the television. For this the object 'remote' invokes the function of the object 'television'.

2.7-2.8 Check your progress

Fill in the blanks

1. ………………… means the ability to take more than one form.
2. ……………… may exhibit different behavior in different instances depending upon the data used in the operation. This is function polymorphism.
3. Operator polymorphism is the process of making an operator to exhibit ………………… in different instances.
4. A ………………… is a request for execution of a procedure that invokes

2.9 SUMMARY

In this chapter the object oriented concepts are reviewed. The most important elements of the object oriented method i.e. classes and objects are considered. The other important features viz. encapsulation, polymorphism, inheritance, message passing are also described.

2.10 CHECK YOUR PROGRESS - ANSWERS

2.2-2.3

1. Identity
2. Operations
3. Structurally identical
4. Instances

2.4 - 2.6

1. False
2. True
3. True
4. True

2.7-2.8

1. Polymorphism
2. An operation
3. different behaviors
4. message

2.11 QUESTIONS FOR SELF – STUDY

1. What is an object?
2. What are classes?
3. What is abstraction?
4. What is inheritance?
5. What is encapsulation?
6. What is the purpose of message passing?

2.12 SUGGESTED READINGS

1. *Unified Modeling Language User Guide by* Grady Booch,James Rumbaugh, Ivar Jacobson
2. *UML 2 For Dummies* by Michael Jesse Chonoles, James A. Schardt

***

NOTES

NOTES

Chapter 3

Overview of UML

3.0 OBJECTIVES

To understand the building blocks of UML

- Things and its four types
- Relationships and its four types
- Diagrams and its three types.

3.1 INTRODUCTION

UML is constructed from three main building blocks: things, relationships and diagrams. When these building blocks are understood correctly it is easy to use the UML.

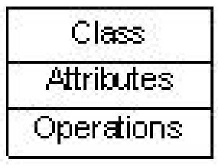
3.2 THINGS

There are four types of things in the UML. These are structural, behavioral, grouping and annotational.

3.2.1 Structural Things

These are also called the nouns of the UML models. These are usually the static parts of the system in question. They represent physical and conceptual elements. Following structural things are available with the UML for use.

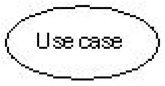
Class – an abstraction of a set of things in the system that have similar properties and/or functionality.



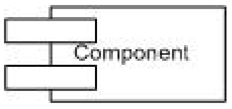
Interface – interface defines a set of operations which specifies what a class can do.

Collaboration – collaboration defines an interaction between things to achieve some goal.

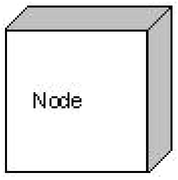
Use case – use case represents a set of actions performed by a system for a specific goal.



Component – component describes a physical part of the system.



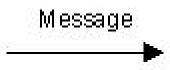
Node – a node can be defined as a physical element that exists at runtime.



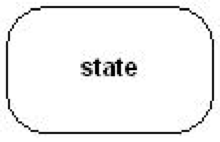
3.2.2 Behavioral things

These are also called the verbs of the UML models. These are usually the dynamic parts of the system. Following behavioral things are available with the UML for use.

Interaction – some behavior constituted by messages exchanged among objects. The exchange of messages is with a view to achieving some purpose.



State machine – state machine is useful when the state of an object in its lifecycle is important. It defines the sequence of states an object goes through in response to events. Events are external factors responsible for state change

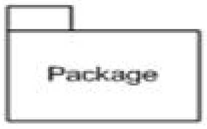


.

3.2.3 Grouping things

This is the organizational part of the UML. It provides a higher level of abstraction. Following grouping mechanism is available with the UML for use.

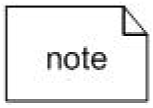
Package – a general purpose element that comprises UML elements – structural, behavioral and even grouping things. These are conceptual groupings of the system and need not necessarily be implemented as cohesive software modules.



3.2.4 Annotational things

This is the explanatory part of the UML model. It adds information or meaning to the model elements. Following annotational mechanism is available with the UML for use.

Note – it is a graphical notation for attaching constraints, rules, comments to elements of the model.



3.2 Check your progress

Fill in the blanks

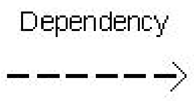
1. ………………….. are called the nouns of the UML models.
2. The behavioral things are called the ……………………. of the UML.
3. ……………………….. are conceptual groupings of the system.

3.3 RELATIONSHIPS

The objects need to interact with each other. The relationships describe the meaning of the links between things and how the links are to be interpreted while implementing the system. There are four types of relationships viz. dependency, association, generalization, realization.

3.3.1 Dependency

Dependency is a semantic relationship where a change in one thing (the independent thing) causes a change in the semantics of the other thing (the dependent thing). The notation used for dependency is a dashed line with an arrowhead.

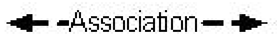


For example,

When the date of birth (the independent thing) is changed, the age (the dependent thing) changes with it.

3.3.2 Association

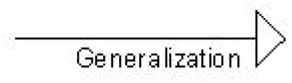
Association is a structural relationship that describes the connection between two things. It also describes how many objects are taking part in that relationship. The notation used for association is a simple line segment or a line segment with arrowhead.



For example,

3.3.3 Generalization

Generalization is a relationship between a general thing (a parent or superclass) and a more specific kind of that thing (a child or subclass). The notation used for generalization is a line segment with an empty block triangular arrowhead. The arrowhead points toward the generalized class or use case or package.

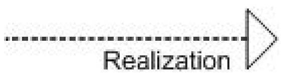


For example,

An animal (generalization) and a cat (specialization) are related by the generalization-specialization relationship.

3.3.4 Realization

Realization is a semantic relationship between two things where one specifies the behavior to be carried out and the other carries out the behavior. I.e. one element describes some responsibility which is not implemented and the other one implements it. This relationship exists in case of interfaces. The notation used is a dashed line with a empty triangular block arrowhead.



For example,

The television carries out the behavior specified by the remote.

3.3 Check your progress

True or false

1. Dependency is a semantic relationship where a change in one thing causes a change in the semantics of the other thing.
2. Association describes how many objects are taking part in that relationship.
3. In the generalization relationship, arrowhead points toward the specialized class

3.4 DIAGRAMS

The diagrams are graphical representation of the models which uses various symbols and text. Each different symbol has a special meaning in the context of the UML. The user can use text to describe the concepts from the system under development. There are thirteen different types of diagrams available in the UML 2.0. Each diagram has its own set of symbols. Each diagram captures a different dimension, view, perspective of the system.

There two major types of the UML diagrams: structure and behavior. The behavior diagrams also include the interaction diagrams.

3.4.1 Structure diagrams

This is a set of diagrams that depicts the elements of a specification those are irrespective of time. This includes class, composite structure, component, deployment, object and package diagrams. These models represent the framework for the system.

3.4.2 Behavior diagrams

This is a set of diagrams that depicts behavioral features of a system or business process. This includes activity, state machine, use case diagrams and the four interaction diagrams. These models describe the interaction in the system.

3.4.3 Interaction diagrams

This is a set of diagrams which emphasize object interactions. The interaction is very important to get something done. This diagram type is categorized under behavior diagram. This includes communication, interaction overview, sequence and timing diagrams.

3.4 Check your progress

Fill in the blanks

1. Structure diagrams is a set of diagrams that depicts the elements of a specification those are ……………….. of time.
2. ………………….. is a set of diagrams that depicts behavioral features of a system or business process.
3. ………………… is a set of diagrams which emphasize object interactions**.**

3.5 SUMMARY

In this chapter we learnt the building blocks of the UML. There are three main building blocks viz. things, relationships and diagrams. The types of diagrams are mentioned here but the different diagrams available are described in the next chapter.

3.6 CHECK YOUR PROGRESS - *ANSWERS*

3.2

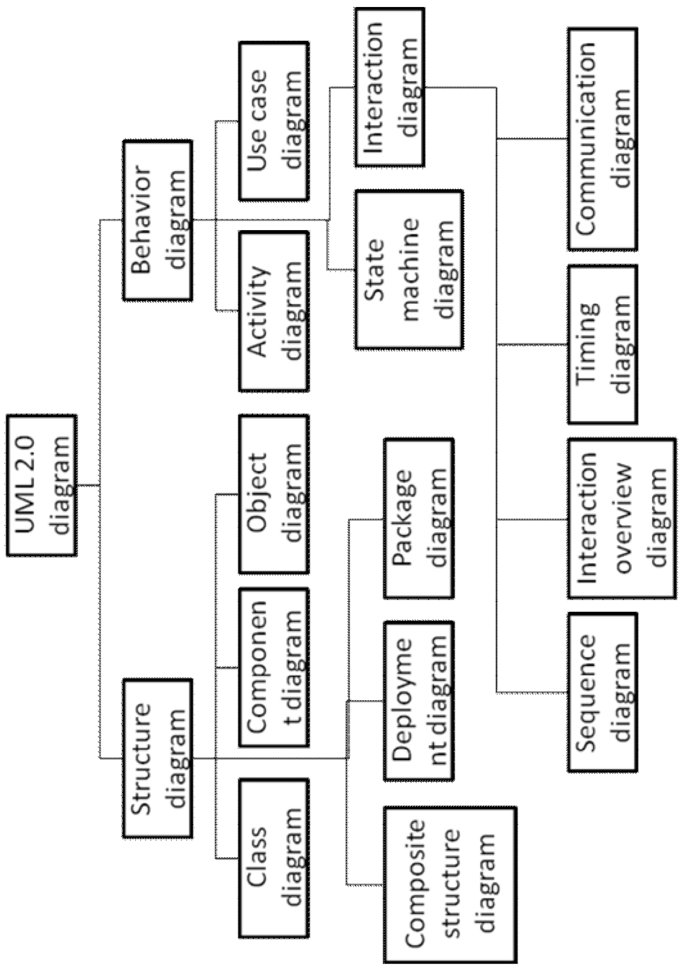
1. Structural things
2. Verbs
3. Packages
4. Annotational things

3.3

1. True
2. True
3. False
4. True

3.4

1. Irrespective
2. Behavior diagrams
3. Interaction diagrams



3.7 QUESTIONS FOR SELF - STUDY

1. Why is it necessary to have a variety of diagrams?
2. Which diagrams give the static view of the system?
3. Which diagrams give the behavioral view of the system?
4. Name major building blocks of UML?
5. What are the different types of UML diagrams?
6. What is interaction diagram?

3.8 SUGGESTED READINGS

1. *Unified Modeling Language User Guide by* Grady Booch,James Rumbaugh, Ivar Jacobson
2. *UML 2 For Dummies* by Michael Jesse Chonoles, James A. Schardt

***

NOTES

NOTES

Chapter 4

Overview of UML Diagrams

4.0 OBJECTIVES

To study the thirteen different diagrams

- Activity diagrams
- Class diagrams
- Communication diagrams
- Component diagrams
- Composite structure diagrams
- Deployment diagrams
- Interaction overview diagrams
- Object diagrams
- Package diagrams
- Sequence diagrams
- State machine diagrams
- Timing diagrams
- Use case diagrams

4.1 INTRODUCTION

There are thirteen different diagrams defined by the UML 2.0. The structure diagrams include class, composite structure, component, deployment, object and package diagrams. The behavior diagrams include activity, state machine, use case diagrams and the four interaction diagrams. The interaction diagrams include communication, interaction overview, sequence and timing diagrams.

4.2 ACTIVITY DIAGRAM

This diagram helps describe the flow of control of the target system, such as the exploring complex business rules and operations, describing the use case as well as the business process. This is the object oriented equivalent of flow charts and data flow diagrams. This is a type of behavior diagram.

This diagram will be explained in detail in a later chapter.

4.3 CLASS DIAGRAM

This diagram shows the classes of the system, their interrelationships and the operations and attributes of the classes. This is a structure diagram.

This diagram will be explained in detail in a later chapter.

4.4 COMMUNICATION DIAGRAM

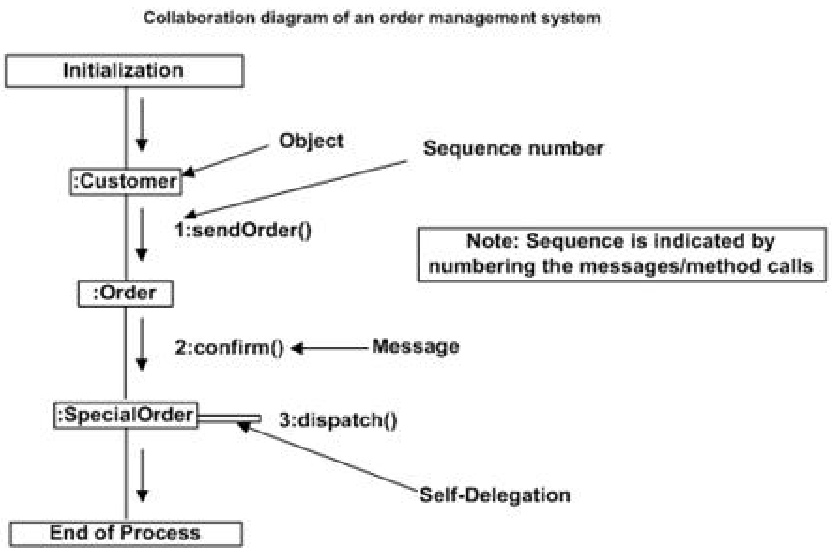
This diagram is used to model the dynamic behavior of the use case. This diagram is more focused on showing the collaboration of objects. This is a type of interaction diagram. This diagram is similar to the sequence diagram. This diagram can be converted into a sequence diagram. They present the same information so it is easy to turn one diagram into the other. This diagram emphasizes the context and overall organization of the objects that interact. This diagram is arranged according the space. The communication diagram is an extension of an object diagram.

This diagram shows in addition to the links between the objects, the messages that are exchanged between them. An arrow is drawn near the link between two objects indicating the message. The arrow points toward the receiving object. The text written near the arrow specifies what the message is. The message tells the receiving object to execute one of its i.e. the receiving object's operations.

To help conversion to a sequence diagram, a number is added to the text of the message, with the number corresponding to the message's order in the sequence. A colon is used to separate the number from the message.

Some of the messages passed can request an object to perform some operation and return the result. In this case the name of the returned value is written on the left, followed by := followed by the name of the operation and the parameters that operation requires to calculate the result.

The conditions are also represented in this diagram. This is done by putting the condition in square bracket.



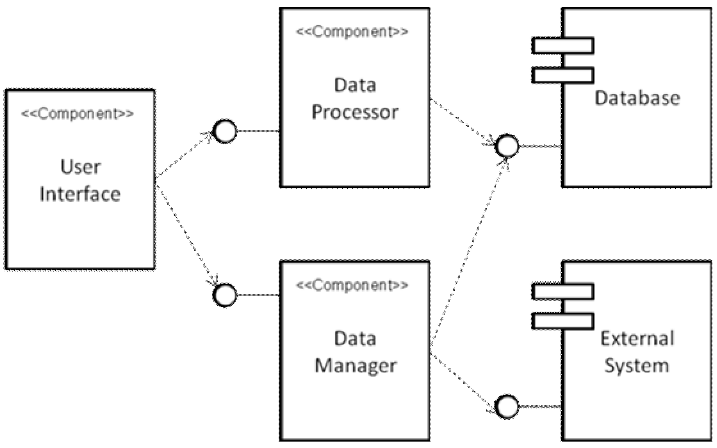
\

4.5 COMPONENT DIAGRAM

This diagram shows the dependencies and the interrelationships among software components and the artifacts that implement them, such as source code files, binary code files, executable files, scripts, and tables. This is a structure diagram. This diagram helps in the management of the system. Software artifacts (classes, interfaces etc.) of a system are arranged in different groups depending upon their relationships during design phase. These groups are known as components. This diagram is used to visualize the implementation.

A software component is a modular part of a system. It consists of one or more classes. A component provides interfaces to other components. A component defines system's functionality. The component hides what it does behind an interface. Other objects ask component to execute its operations. The interface is a set of operations that a component presents to other objects.

A component diagram contains components along with the interfaces and relationships. The component is represented as a rectangle with a keyword <<component>> near the top in UML 2.0. It also can be shown in another manner as a rectangle with two small rectangles overlaid on its sides as was popular in the prior versions of UML. A line with a circle at one end represents the interface through which the components communicate with each other. This communication is shown through a relationship depicted by a dotted line with an arrowhead toward the interface.



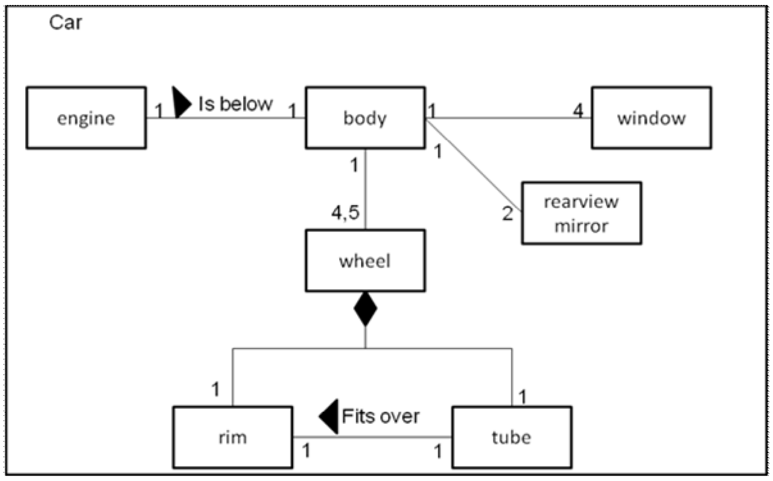
4.2-4.5 Check your progress

Fill in the blanks

1. ………………….. is the object oriented equivalent of flow charts and data flow diagrams.
2. Communication diagram is used to model the ……………….of the use case.
3. A ………………….is a modular part of a system which consists of one or more classes.

4.6 COMPOSITE STRUCTURE DIAGRAM

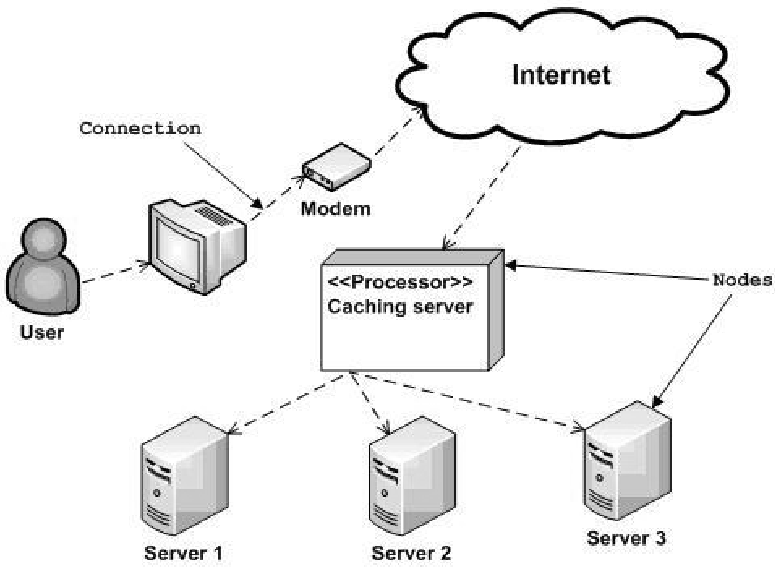
This diagram is used to explore run-time instances of interconnected instances collaborating over communication links. It shows internal structure of collaboration. This is a structure diagram. The composite is one method used to show the components of a class. This diagram visualizes the internal structure of a class by showing classes nested inside that class.



4.7 DEPLOYMENT DIAGRAM

This diagram depicts the static view of the run-time configuration of hardware nodes and the software components that run on those nodes. It shows the hardware of the system, the software that is installed on that hardware and the middleware that is used to connect the dissimilar machines to one another. These nodes are physical entities where the components are deployed. This diagram is used to visualize the deployment view of a system. This is a structure diagram. This diagram helps in the management of the system.

Deployment diagram of an order management system



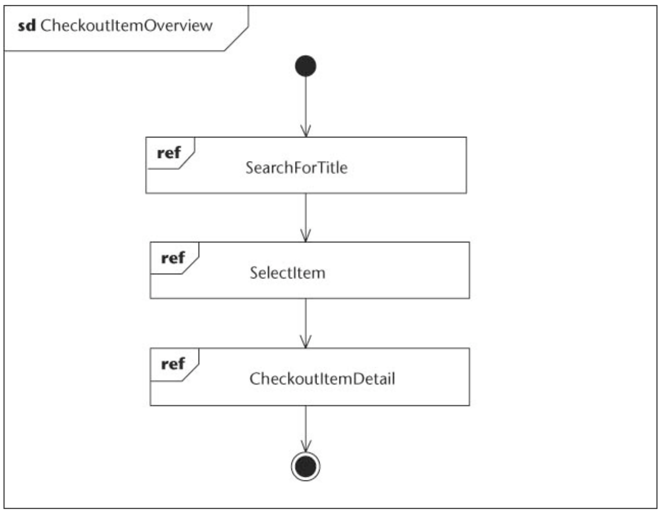
4.8 INTERACTION OVERVIEW DIAGRAM

This diagram focuses on the overview of the flow of control of the interactions. It is a variant of the activity diagram where the nodes are the interactions or interaction occurrences. This is a type of interaction diagram.

Some of the use cases may involve different complex scenarios one of them being the success scenario and error flows. This kind of situation may need knowledge or the reference of another use case or scenario. This diagram helps us solve this problem of referencing other scenarios to simplify the complexity.

In this diagram, the interaction occurrence nodes are connected by control flows instead of showing with the help of activity nodes and actions. This interaction occurrence is nothing but a representation of a full sequence diagram. Wherever this is present in the diagram, it references the corresponding sequence diagram.

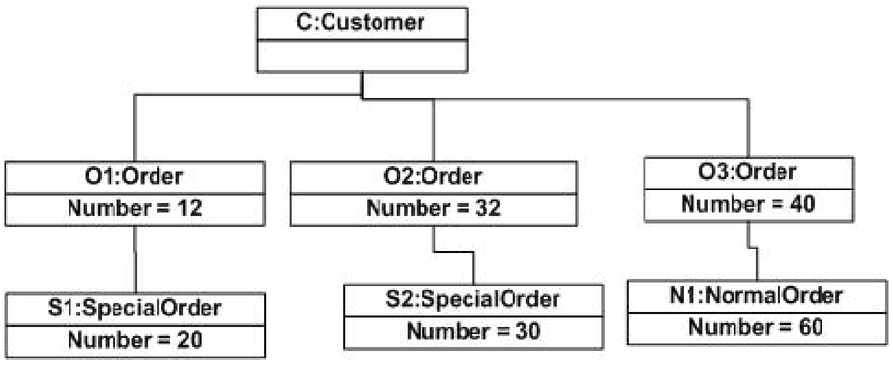
The interaction occurrence is denoted by a rectangle. The top left corner of the rectangle has a smaller rectangle with the bottom right corner removed and the keyword ref written inside it. The name of the referred use case is written inside the rest of the bigger rectangle.



4.9 OBJECT DIAGRAM

This diagram is useful for exploring real world example of objects and the relationships between them. This is a variant of the class diagram where instances are shown instead of classes. So this is also known as instance diagram. This diagram is more close to real life scenarios where we implement a system. This diagram is a set of objects and their relationships. This diagram is useful for explaining small pieces with complicated relationships, especially recursive relationships. This is a structure diagram. The object diagrams are used in similar way to the class diagrams but they are used to build prototype of a system from practical perspective.

Object diagram of an order management system



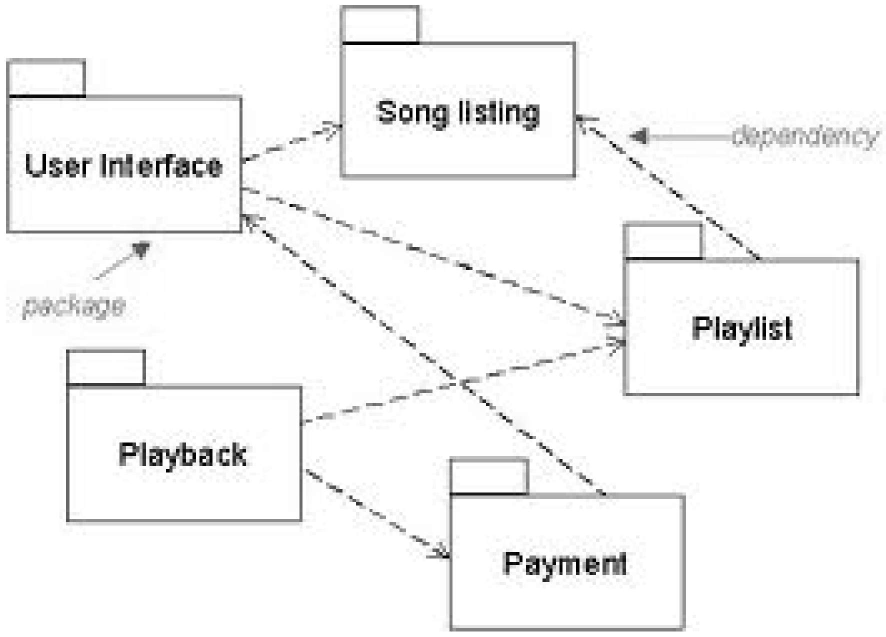
4.6-4.9 Check your progress

True or false

1. Composite structure diagram shows internal structure of collaboration.
2. The nodes in deployment diagram are virtual entities where the components are deployed.
3. Interaction overview diagram is a type of structure diagram.
4. Object diagram is useful for explaining small pieces with complicated relationships, especially recursive relationships.

4.10 PACKAGE DIAGRAM

This diagram simplifies complex class diagrams by grouping classes into packages. A package is a collection of logically related UML elements. Packages are depicted as file folders and can be used in any of the UML diagrams. This is a structure diagram. This diagram helps in the management of the system.



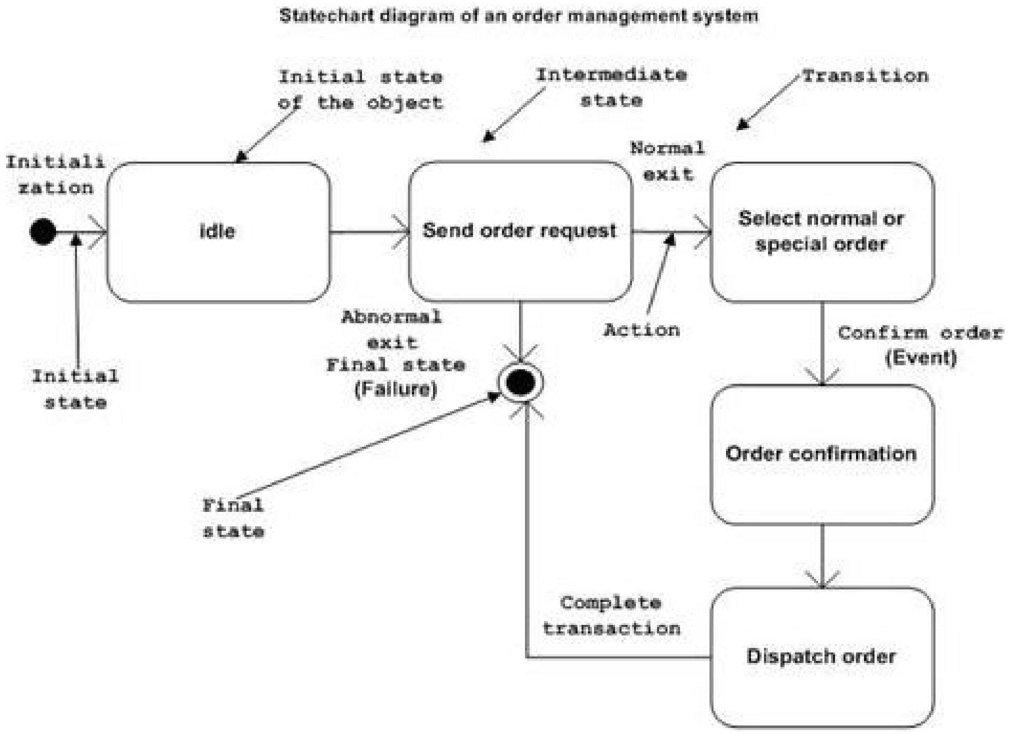
4.11 SEQUENCE DIAGRAM

This diagram models the collaboration of objects based on a time sequence. This shows how the objects interact with others in a particular scenario. This type of diagram deals with the sequences of messages flowing from object to another. This diagram depicts the interaction among the components of a system which is very important from implementation and execution point of view. This diagram is used to visualize the sequence of calls in a system to perform a specific functionality. This is a type of interaction diagram.

This diagram will be explained in detail in a later chapter.

4.12 STATE MACHINE DIAGRAM

Any real time system and its elements are expected to react to some kind of external/internal events. These events are responsible for the state change of the system and/or one/more elements within the system. This diagram can show the different states of an entity. This also shows how an entity responds to various events by changing from one state to another. This diagram is used to visualize the reaction of a system and/or its elements by internal/external factor. This helps model the history of an entity. This is a type of behavior diagram.

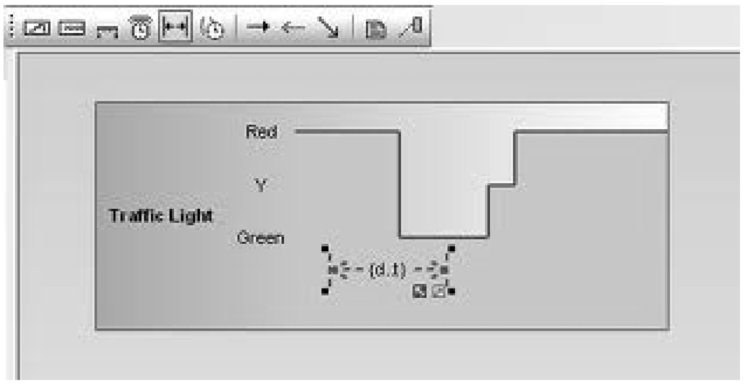


4.13 TIMING DIAGRAM

This diagram shows the behavior of the objects in a given period of time. This is a special form of a sequence diagram. This is a type of interaction diagram. These diagrams are used only in the systems which are time critical i.e. if there is a delay of even one second the behavior of the system can be affected.

For example,

In case of a nuclear reactor, the slightest delay is likely to cause very large scale destruction. Or if the signals do not change on time there is a chance of an accident in the railway network.



4.14 USE CASE DIAGRAM

This diagram describes the behavior of the target system from an external point of view. This describes the actual functional requirements. Use cases, actors and their relationships. a use case represents a particular functionality of a system. This diagram is used to describe the relationships among functionalities and their internal/external controllers. They represent the use case view of a system. This is a type of behavior diagram.

This diagram will be explained in detail in a later chapter.

4.10-4.14 Check your progress

Fill in the blanks

1. ………………. simplifies complex class diagrams by grouping classes into packages.
2. ………….. diagram is used to visualize the reaction of a system and/or its elements by internal/external factor.
3. Timing diagram is a special form of a……………..

4.15 SUMMARY

There are thirteen different diagrams available with the UML. Each diagram gives a different view of the system under construction.

Source: *www.objectmentor.com(Link)*

4.16 CHECK YOUR PROGRESS - *ANSWERS*

4.2-4.5

1. Activity diagram
2. dynamic behavior
3. software component

4.6-4.9

1. True
2. False
3. False
4. True

4.10-4.14

1. Package diagrams
2. State machine
3. sequence diagram

4.17 QUESTIONS FOR SELF - STUDY

1. What are the main usages of a deployment diagram?
2. What are the usages of sequence and collaboration diagram?
3. What is a Statechart diagram?
4. What does an object diagram capture?
5. What is a node?
6. What is component diagram?
7. What is a component?
8. What is the difference between a class diagram and an object diagram?
9. What is the difference between sequence and collaboration diagram?
10. What is the main usage of a Statechart diagram?

4.18 SUGGESTED READINGS

1. *Unified Modeling Language User Guide by* Grady Booch,James Rumbaugh, Ivar Jacobson
2. *UML 2 For Dummies* by Michael Jesse Chonoles, James A. Schardt

***

NOTES

NOTES

Chapter 5

Use Case Diagrams

5.0 OBJECTIVES

- To understand the need for use cases
- To study different model elements used in the use case diagram
  - Actors
  - Use cases
  - Relationships

5.1 INTRODUCTION

The most important phase in the software development life cycle is the requirements gathering stage. If the requirements are gathered collected correctly and unambiguously, the developed system will have high chances of being accepted. The use case diagrams help in gathering the requirements.

5.2 NEED FOR USE CASES

When we are dealing with project development, there is a need to capture the requirements correct, unambiguously. The requirement gathering phase in the software development phase allows us to collect requirements. The requirement collection defines the functionality to be provided and identifies the goals to be achieved. The requirements must be precisely and completely understood by the team providing the solution. User requirements keep changing, so the requirements must be well-documented. A thorough and unambiguous understanding of the requirements is vital to ensure that everyone knows what they are doing and why. The requirements must be reviewed again and again before the design and implementation begins. This phase involves the participation of the domain experts to ensure that the requirements have been correctly understood. This phase captures the ‘WHAT’ of the problem.

Use cases are used to capture, describe and document the requirements. The use case diagram gives the idea of what the entire system is required to do. It is very necessary to get the use cases correctly since we will be referring back to the use case diagram throughout the life cycle of project development.

5.2 Check your progress

True or false

1. Requirement gathering phase captures the 'WHAT' of the problem.
2. The requirements need not be precisely and completely understood.

5.3 DIAGRAM MODEL ELEMENTS

There are three elements used in the use case model.

- Actors
- Use cases
- Relationships

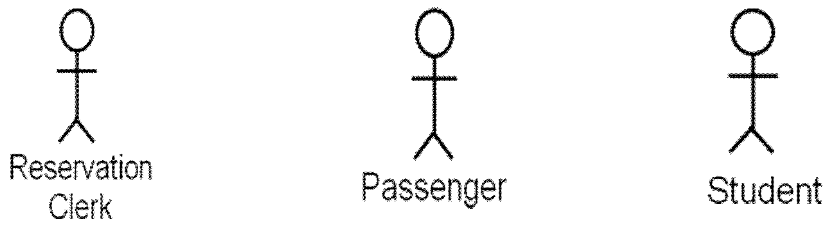
The relationships are of three types

- Between actors and use cases
- Between use cases
- Between actors

5.3.1 Actors

An actor represents a role that interacts with the system. It represents a role, not an individual. A person, a device or another system can be an actor. The communication of the actor with the system is established through sending and/or receiving messages. An actor may participate in many use cases and a use case may have several actors participating in it.

The notation used for a use case is a stick person or a node or a device symbols and the role name is written below it. To simplify the matter one can simply use a role written inside << >> to specify the actor name.



5.3.2 Finding actors

There are certain points on the basis of which one can think of searching for the actors in the system.

- Who uses the main functionality of the system?

  Reservation clerk, Passenger, Customer, Manager
- Which hardware devices the system needs to handle?

  Printer, Scanner, RFID reader, Barcode reader
- Which other systems does the system need to interact with?

  Paypal, Bank, Salary calculator, Tax calculator
- What nouns are used to describe the system?

  User, Reservation clerk, Administrator

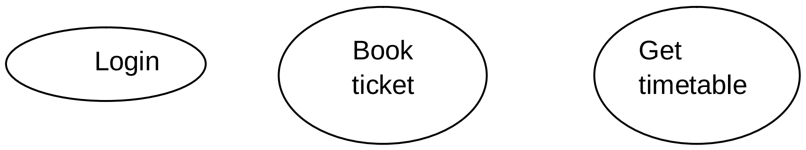
5.3.3 Use cases

This is an abstraction of a set of sequences that yield some functionality. E.g. *login*, *book a ticket*, *purchase item*. *Login* can be correct login or incorrect login where user may have provided the username, password or both incorrectly. So under the use case login both these scenarios will be captured. There is no need to specify two separate use cases login correctly and login incorrectly. As also the three possibilities for incorrect login also will be covered.

Use case represents some user-visible function. Login is a user visible function. It will not describe how actually the login will be done, what steps are involved in it. Use case is always initiated by an actor. It describes the interaction between the actors and the system to get some function performed. This achieves some discrete goal for the actor.

Use cases are narrative descriptions of business processes. A use case represents from start to finish, a sequence of events, actions and transactions required to produce or complete something of value to an actor.

The notation used for a use case is an oval with the use case written inside it. Since use cases are 'WHAT' of the system, the actions performed they are written as verbs or verb phrases. So ticket will not be a use case whereas *print ticket*, *book ticket*, *get ticket* will be use cases.



5.3.4 Finding use cases

The following points can be considered when looking for the use cases in the system.

- What functions the system is expected to perform?

  Update the database periodically. The system is remotely shut down at 8pm and booted at 6am every day.

- What do the actors expect system to perform?

  Display the train list. Show the fares for different classes of accommodation.

- What are the inputs and outputs expected by the system?

  The system will print the ticket. The customer gets the reservation form. A sales report is generated at the end of the day. A list for items which have reached reorder level is generated and sent to the manager..

- What are the verbs which are used to describe the system?

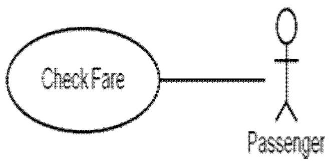
The reservation clerk makes the booking using the system. The railway administrator can add new trains, modify existing train details. A customer can fill in the reservation form.

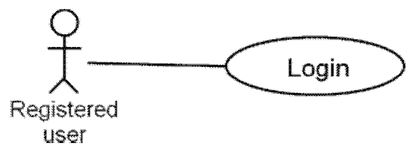
5.3.5 Relationships

5.3.5.1 Actor to use case

Any use case has to be initiated by an actor. All the use cases are not initiated by all the actors. Any actor initiates some of the use cases. To indicate which actor initiates which use case there has to be a relationship between them. This link indicates interaction between a specific actor and a specific use case.

The notation used for this relationship between actor and use case is a plain line between the relevant actor and use case.





5.3.5.2 Between use cases

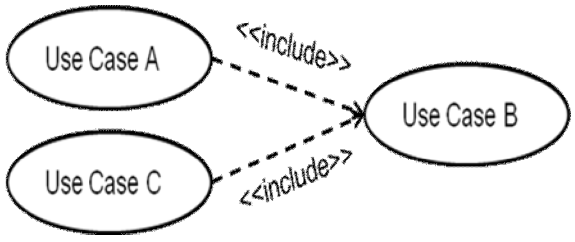
There are two types of relationships that can exist between a pair of use cases. These are

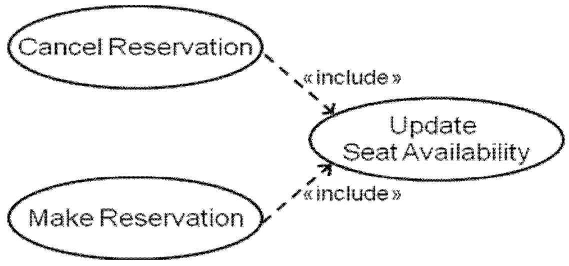
5.3.5.2.1 Use/Include

When some behavior is similar across more than one use case, the common behavior is factored out and represented in a separate use case. The use cases that include the common behavior are related to the use case that represents the common behavior by the uses or includes relationship.

Use case A uses or includes use case B when use case B is a behavior/functionality that is required by use case A. That behavior is factored out into a separate use case because it is required across several use cases.

The notation used for the extend relationship is a dashed arrow with <<include>> or <<use>> written over it. The arrow head is toward the use case representing the common functionality. To decide the direction of the arrowhead a question should be asked "what gets included?" and put the arrow head along that use case.



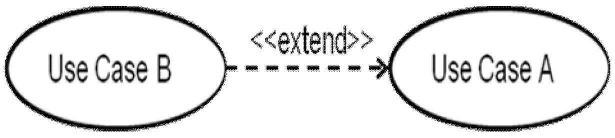


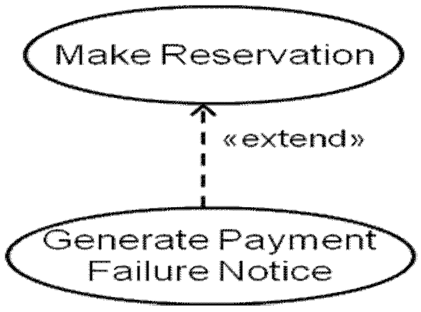
5.3.5.2.2 Extend

This relationship is used to relate use cases which describe variations in the normal flow of events for a particular function to the basic use case which describes the normal flow of events for the function. An extending use case is used to describe variations in the normal flow of events described by a general use case.

Use case B extends use case A when use case B describes the behavior of use case A under a particular condition.

The notation used for the extend relationship is a dashed arrow with <<extend>> written over it. The arrow head is toward the use case representing the normal flow. To decide the direction of the arrowhead a question should be asked "what gets extended?" and put the arrow head along that use case.

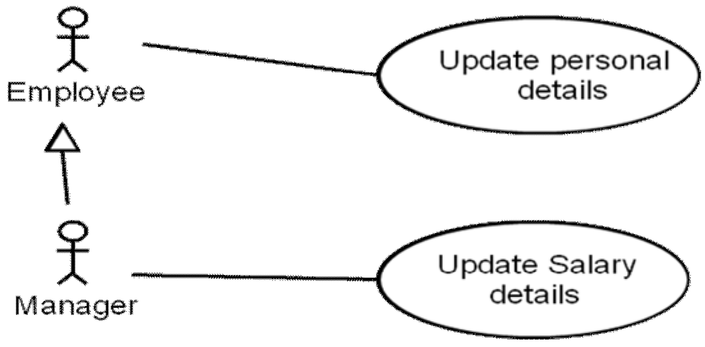




5.3.5.3 Between actors

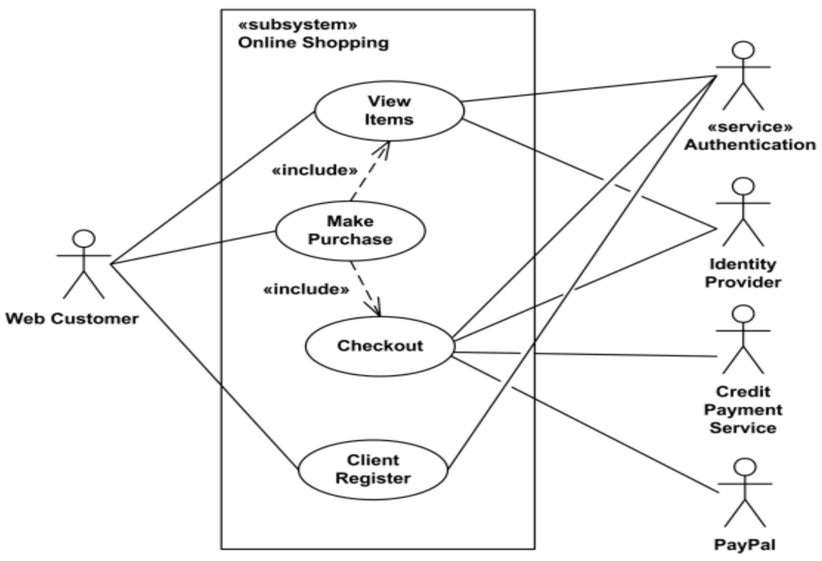
Two actors can be related only through the generalization-specialization relationship. This is used if the specialized actor is doing some kind of interaction which is not done by the rest of the actors who are involved in this relationship. If there are no additional special functionalities to an actor it is recommended not to use this relationship just for the sake of using it.

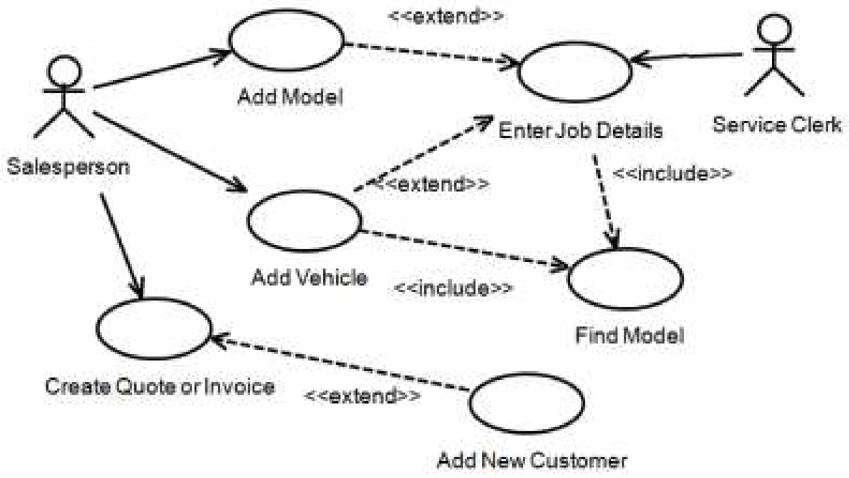
The notation used a line with arrow head where the arrow head is empty triangular head. The head points toward the generalized actor.



5.4 USE CASE DIAGRAMS

The use case diagram is a graphical representation of the use cases of a system, its actors and the interactions between them. It also depicts the system boundary – what is included in the system, what is not included in the system, what is outside the system and interacts with the system. It is constructed from the use cases, actors and the relationships between them. A system boundary is optionally shown by drawing a rectangular box. The use cases are placed inside the box and the actors are placed outside the box. The interaction between the actor and use case is shown by a line intersecting the side of the box.





5.3 **- 5.4 Check your progress**

Fill in the blanks

1. An actor represents a …………….. that interacts with the system.
2. ……………… is an abstraction of a set of sequences that yield some functionality.
3. When some behavior is similar across more than one use case, the common behavior is factored out and represented in a separate use case. This is represented by ……………… relationship.
4. An extending use case is used to describe ……………… in the normal flow of events described by a general use case.
5. The use case diagram is a ……………… representation of the use cases of a system, its actors and the interactions between them.

5.5 SUMMARY

When we wish to start developing a system there is a need to gather the requirements of the system properly. The use case diagram helps in the requirements gathering phase. There are three main elements of the use case diagrams viz. actors, use cases and relationships.

5.6 CHECK YOUR PROGRESS - ANSWERS

5.2

1. True
2. False

5.3-5.4

1. Role
2. Use case
3. Include
4. Variations
5. graphical

5.7 QUESTIONS FOR SELF - STUDY

1. Describe the purpose of use case diagram?
2. What is the purpose of an actor?
3. When is the include relationship used?
4. When is the extend relationship used?
5. Which kind of relationship is possible for actors? When is it used?
6. What is a use case and when is it used?
7. Create a use case diagram for a library system. State your assumptions.

5.8 SUGGESTED READINGS

1. *Unified Modeling Language User Guide by* Grady Booch,James Rumbaugh, Ivar Jacobson
2. *UML 2 For Dummies* by Michael Jesse Chonoles, James A. Schardt

***

NOTES

Chapter 6

Activity Diagram

6.0 OBJECTIVES

- To understand the activity diagrams
- To study the different elements of the activity diagrams
  - Start symbol
  - End symbol
  - Activity
  - Forks and joins
  - Decision points / branch
  - Merges
  - Guard / condition
  - Swimlane / partition
  - Object node
  - Transition / control flow

6.1 INTRODUCTION

There are certain parts in the system which is very complex. The activity diagrams describe the flow of control from activity to activity. This helps simplify the part involving complexity.

6.2 NEED FOR ACTIVITY DIAGRAM

The activity diagram is a diagram that models business processes. It also emphasizes the flow of control from activity to activity. It is very much similar to the traditional program flowchart. It is typically used to provide details for complex algorithms. This diagram is typically used for business process modeling, for modeling the logic captured by a single use case or usage scenario.

6.3 CREATING ACTIVITY DIAGRAM

While creating an activity diagram that following steps could be taken.

- Set the context or scope of the activity being modeled.
- Give the diagram an appropriate title.
- Identify the activities, control flows and object flows that occur between the activities.
- Identify any decisions that are part of the process being modeled.
- Identify any prospects for parallelism in the process.
- Draw the activity diagram.

6.2-6.3 Check your progress

True or false

1. The activity diagram is a diagram that models business processes.
2. While creating activity diagram there is no need to set the context.

6.4 WHEN TO USE ACTIVITY DIAGRAMS

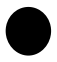
The diagrams can be used

- To analyze a use case
- To understand workflow within the system
- To describe a complicated algorithm
- To model multithreaded or parallel behavior

6.5 ELEMENTS OF THE ACTIVITY DIAGRAMS

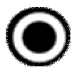
6.5.1 Start symbol

This indicates the starting point of the diagrams. This is optional. If it is used it makes it significantly easier to read the diagram. This is denoted by a filled in circle.



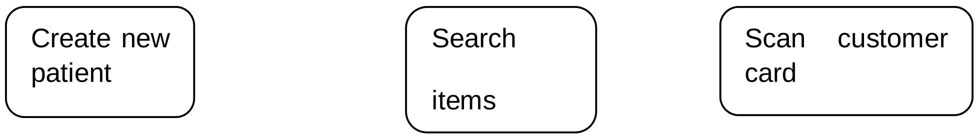
6.5.2 End symbol

This indicates the final node after which there will not be any more activities. This is denoted by a filled in circle with a boundary.



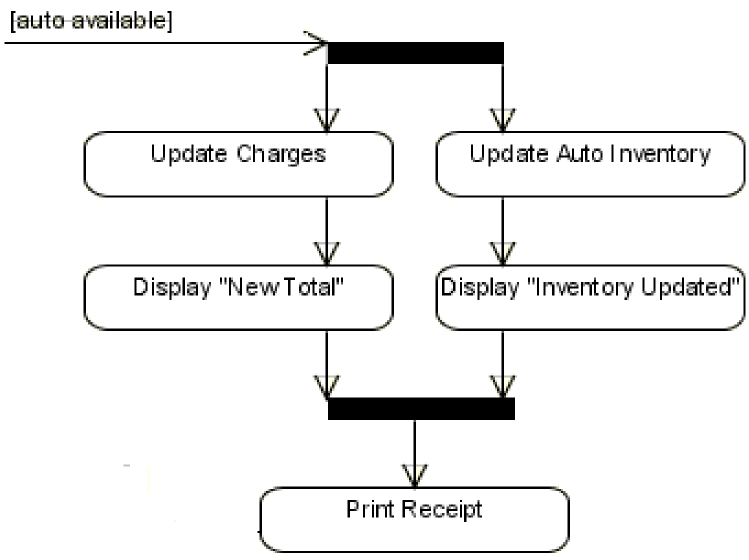
6.5.3 Activity

This is a behavior an object carries out while in a particular state. A rounded rectangle specifies an activity. The name of the activity is to be written inside the rounded rectangle.



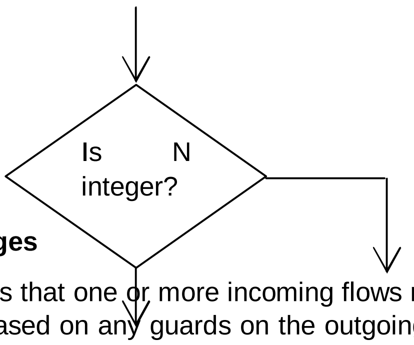
6.5.4 Forks and joins

These are used when we wish to denote the parallel or concurrent paths of activities. When the concurrent or parallel paths begin at the top we denote a thick line called the synchronization bar (fork) and have different parallel paths coming from it shown by transitions. Once all the concurrent or parallel activities are over they all should come together to continue with the further processing. Here we have multiple transitions coming together in a thick line called the synchronization bar (join).



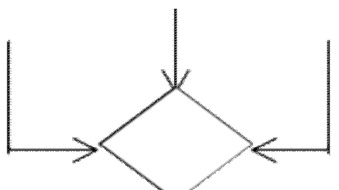
6.5.5 Decision point/branch

When we find a situation with different possibilities then we need to decide which path is to be taken. To indicate the choice decision points are used. This is denoted by a diamond symbol containing a condition whose results provide transition to different paths. Typically the condition to be checked is written inside the diamond. There is more than one possible path to choose from. During execution one of the possible paths will be selected. There one transition entering and several transitions leaving the diamond. The outgoing flows include conditions on the basis of which that particular transition is selected.



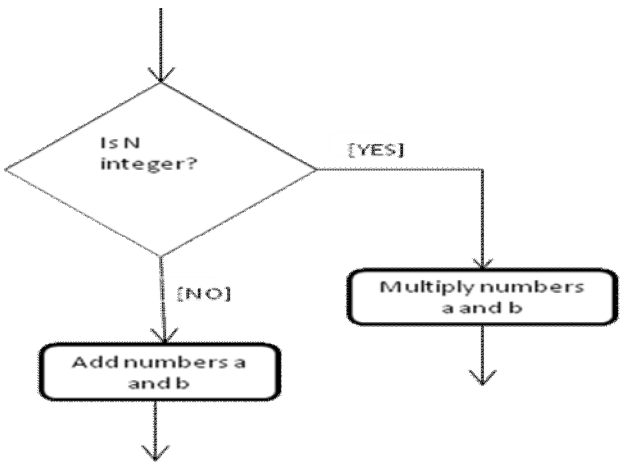
6.5.6 Merges

This indicates that one or more incoming flows must reach this point until processing continues, based on any guards on the outgoing flow. This is denoted by a diamond with several flows entering and one leaving.



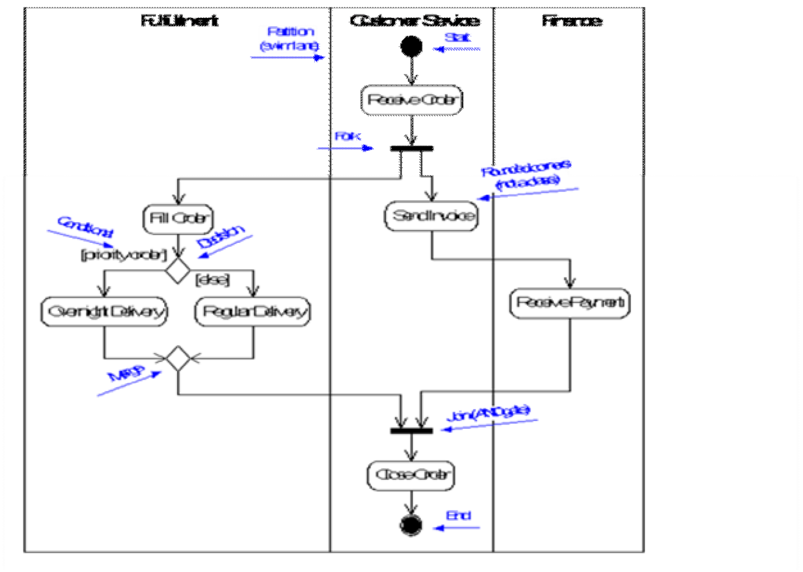
6.5.7 Guard/condition

This is given by text written in between '[' and ']'. This is placed on a flow. The guard must evaluate to TRUE in order to traverse the node.



6.5.8 Swimlane/partition

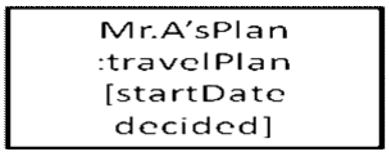
The swimlanes or partitions indicate who or what is performing the activities. Every participant gets a swimlane in which that participant's activities are shown. The interactions are shown by transitions across swimlanes.

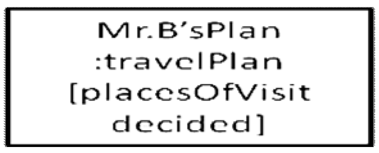


6.5.9 Object node

The classes have operations which take in parameter and generate results. These operations may modify one or more of the attributes of the class. Thus the state of the object changes. Activities also may modify objects or they may transform one object to another.

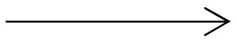
This is shown by a rectangle inside which the name of the object is written. Below the name inside square brackets the state of the object is mentioned.





6.5.10 Transition/control flow

This indicates the movement from one activity to another. The arrowhead indicates the destination activity. This is shown by a line with an arrowhead.



6.4 Check your progress

Fill in the blanks

1. ………………. is a behavior an object carries out while in a particular state.
2. ………………. are used when we wish to denote the parallel or concurrent paths of activities.
3. To indicate the choice, …………………are used.
4. The …………….. or partitions indicate who or what is performing the activities.

6.6 SUMMARY

The activity diagram is used to understand the flow of the business functions. This helps in explaining the parts which are complex to understand. There are a variety of elements used in these diagrams.

6.7 CHECK YOUR PROGRESS – *ANSWERS*

6.2-6.3
1. True
2. False

6.4
1. Activity
2. Fork and join
3. decision points
4. swimlanes

6.8 QUESTIONS FOR SELF - STUDY

1. Must every activity diagram have an initial state?
2. How are decisions denoted in an activity diagram?
3. Define the terms transition, activity and fork-and-join.
4. When can fork-and-join be used?
5. What are guards/conditions?
6. Design an activity diagram for purchasing a book online. State your assumptions.

6.9 SUGGESTED READINGS

1. *Unified Modeling Language User Guide by* Grady Booch,James Rumbaugh, Ivar Jacobson.
2. *UML 2 For Dummies* by Michael Jesse Chonoles, James A. Schardt

***

NOTES

Chapter 7

Sequence Diagrams

7.0 OBJECTIVES

- To understand the need for sequence diagrams
- To know the elements involved in the sequence diagrams

▫ Object
▫ Lifeline
▫ Synchronous message
▫ Asynchronous message
▫ Return
▫ Creation of object
▫ Destruction of object
▫ Looping
▫ Boundary
▫ Naming the diagram

7.1 INTRODUCTION

Each use case involves different interactions between a set of objects. These interactions follow a certain sequence to get the job done. The sequence diagram shows the interaction based on the time ordering.

7.2 NEED FOR SEQUENCE DIAGRAM

To understand how the different objects interact with each other, some type of interaction diagram is necessary. If there is a need to understand in what sequence the interaction takes place, the sequence diagram is best suited for the purpose. This diagram is best suited to explore the different scenarios or flows of a particular use case. This diagram also helps understand which classes are needed for the interaction. These diagrams can be easily drawn. They are also easy to understand for the developers and clients. These diagrams basically display their lifelines of participants as they exchange messages. A lifeline shows the life of the participating object by showing the relevant events that are important to the object in that interaction.

7.3 CREATING SEQUENCE DIAGRAM

There is no need to draw sequence diagram for each and every use case and each and every scenario of the use cases.

- All the use cases are to be studied well and then which of them are likely to be complex is to be found out. If the interaction is a typical one, then you may safely ignore that particular use case.
- One by one pick up the use cases which were listed in the above step. Explore all the scenarios of to find out which are not routine ones. Pick up only those which have complex interaction involved.
- Use the same actor from the use case diagram who initiated that use case.
- Find out which other objects need to interact to fulfill the functionality suggested by the chosen use case and its scenarios.
- Decide on the sequence of messages required to be exchanged to fulfill the requisite functionality. Also find out which object is sending the message and which one is receiving as well as what message is passed.
- Determine the type of the interaction i.e. whether it is synchronous, asynchronous, return, creation of object, or destruction of object.
- Draw the objects at the top with the initiating actor at the extreme left.
- Below each of the above objects including the actor draw vertical dashed lines to indicate the lifelines.
- Show interactions between the objects by drawing horizontal arrows between the corresponding lifelines. (To decide the type of arrow consider the type of the interaction determined above)
- Write text along the arrow to indicate the message passed. The message could plain text or it could be function or procedure call.
- Put a box around all these to indicate that this is interaction for one particular use case.
- At the top left corner using a special notation write the name of the corresponding use case.

7.2-7.3 Check your progress

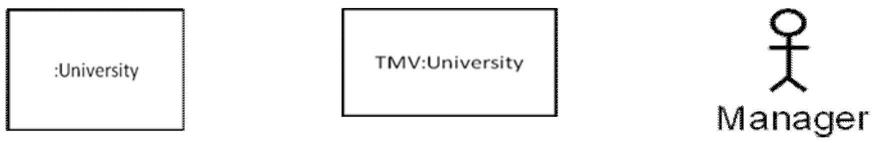
True or false

1. The sequence diagrams basically display their lifelines of participants as they exchange messages.
2. There is a need to draw the sequence diagrams for each and every use case and each and every scenario.

7.4 ELEMENTS OF SEQUENCE DIAGRAM

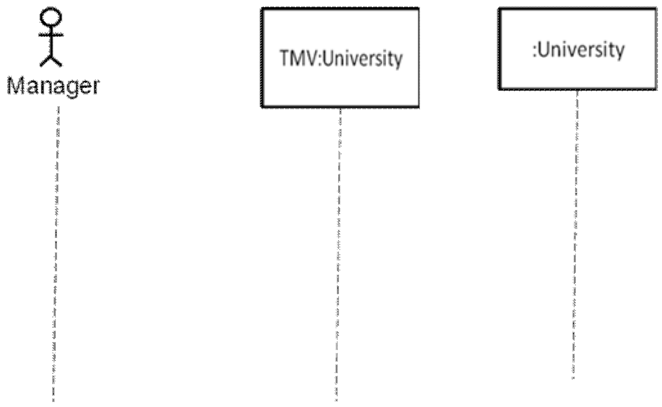
7.4.1 Object

The objects if known or the classes are shown using a rectangular box with name written inside it. If the object is known write inside the box the_object_name: the_class_name to indicate which class the object is an instance of. If the object is not known simply write :the_class_ name. If it is an actor then the stick figure is used.



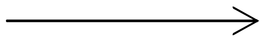
7.4.2 Lifeline

The lifeline is shown vertical with a dashed line starting from the corresponding actor going down till after the last interaction.



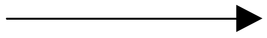
7.4.3 Synchronous message

An object passes a message and waits till it receives some reply from the recipient object. If one object asks another object to calculate salary and it wants to get back the result so that it can print the salary slip, then this type of message is used. It is denoted by a line with a plain arrowhead.



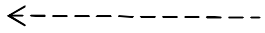
7.4.4 Asynchronous message

When a message is passed by one object to another, and the sender object does not wish to wait till the action is carried out by the other object, this type of message is used. It is denoted by a line with a triangular filled block head.



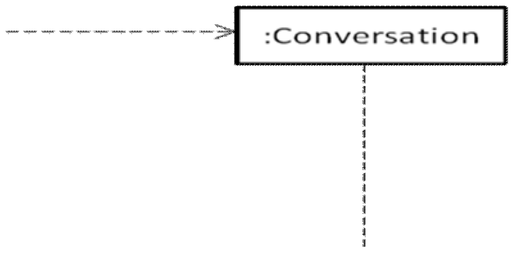
7.4.5 Return

This is used to indicate return from a call. It is not mandatory to show the return arrow. It can be assumed that the recipient carries out its job correctly and if asked returned the requisite information. It is denoted by a dashed line with a plain arrowhead.



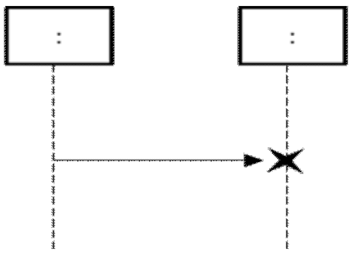
7.4.6 Creation of object

During the interaction some object is not required from the beginning and it has to be created explicitly then message is used. It is shown as given below.



7.4.7 Destruction of object

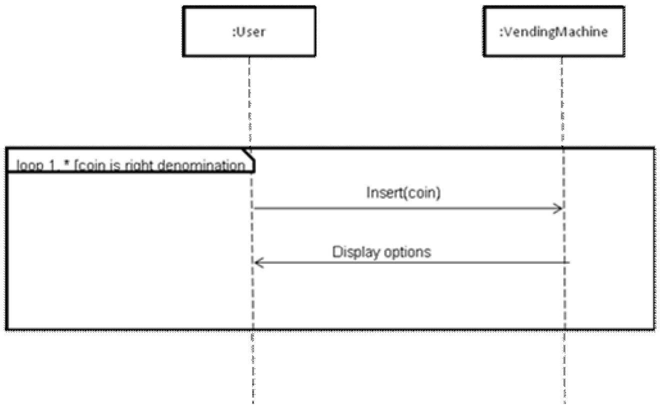
During the interaction some object has to be destroyed explicitly this type of message is used. On the lifeline of the object which is to be destroyed a cross is drawn at the end of the arrow.



7.4.8 Looping

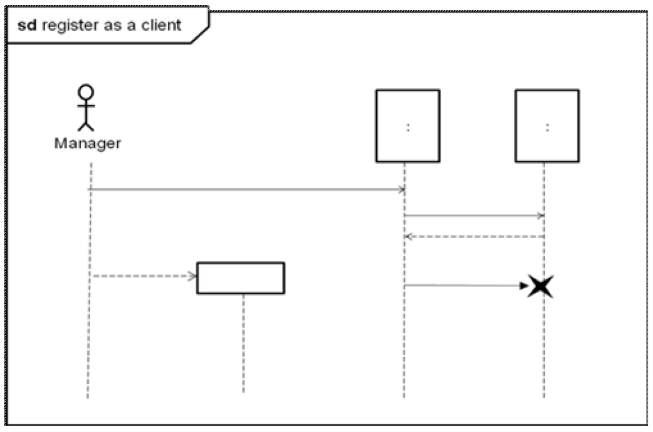
There might be a situation in which there is a need to perform some set of interactions again and again. This is a place which is a situation where a loop is required. Within a sequence diagram all the steps which need to be repeated are picked up. A box put around these steps. In the top left corner in a rectangle with the right bottom corner removed is placed. Inside this corner rectangle, the looping condition is written. The syntax for this is loop minint, maxint,[guard]

This means execute the interaction minint times, then execute interaction up to maxint times as long as the [guard] is true.



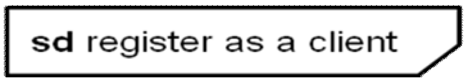
7.4.9 Boundary

The boundary encloses the set of objects, their lifelines, their interactions involved in the sequence diagram drawn for a particular use case. It is a simple rectangle.



7.4.10 Naming the sequence diagram

Every sequence diagram has to be given an identity. This identity is specified by the name of the use case for which the particular sequence diagram is drawn. This is written in a rectangle with the lower right corner cut off.



7.4 Check your progress

Fill in the blanks

1. In a …………………… an object passes a message and waits till it receives some reply from the recipient object.
2. Loop ………………. , ……………….[guard] is the syntax used for looping in a sequence diagram.
3. The ……….. encloses the set of objects, their lifelines, their interactions involved in the sequence diagram drawn for a particular use case.

7.5 SUMMARY

The sequence diagrams are used to clarify the complexity in the execution of any interaction. Here only the complex scenarios for use cases are modeled. This diagram gives the time ordering of the interactions.

7.6 CHECK YOUR PROGRESS – *ANSWERS*

7.2-7.3

1. True
2. False

7.4

1. Synchronous message
2. Minint, maxint
3. Boundary

7.7 QUESTIONS FOR SELF - STUDY

1. What is the purpose of sequence diagrams?
2. What are the different elements used in sequence diagrams?
3. Define synchronous message and asynchronous message.
4. How is a recursion shown in a sequence diagram?
5. How is a newly created object represented in a sequence diagram?
6. Draw a sequence diagram to show the sequence of interactions in the use case 'LOGIN". State your assumptions.

7.8 SUGGESTED READINGS

1. *Unified Modeling Language User Guide by* Grady Booch,James Rumbaugh, Ivar Jacobson
2. *UML 2 For Dummies* by Michael Jesse Chonoles, James A. Schardt

***

NOTES

Chapter 8

Class Diagrams

8.0 OBJECTIVES

- To understand the need for class diagrams
- To know the elements involved in the class diagrams
  - Class
  - Visibility
  - Multiplicity
  - Association
  - Generalization-specialization
  - Dependence
  - Realization
  - Aggregation
  - Composition

8.1 INTRODUCTION

The class diagram shows the different classes involved and the relationships between different classes. The class also shows the details like the attributes and the operations carried out by that class. The relationships describe the details like visibility, multiplicity etc.

8.2 NEED FOR CLASS DIAGRAM

The purpose of a class diagram is to model the static view of an application. This is the only diagram which finds direct relation to object oriented languages and thus can be extensively used at the time of construction. This diagram helps in the analysis and design of the static elements of an application. They describe the responsibilities of the system from the implementation point of view. This diagram can help in the development of the component and deployment diagrams.

This is the most common diagram used in the UML. The class diagram consists of classes, interfaces, relationships and collaborations. This diagram basically represents the object oriented view of a system which is static in nature. It is generally used for development purpose. This is the most widely used diagram at the time of construction of the system.

8.3 CREATING CLASS DIAGRAM

Since this diagram is very much useful in the construction of the application, there is need to study the class diagram in detail.

- Decide which classes, interfaces should be involved.
- Find out the relationships among these elements.
- Identify the responsibilities (attributes and operations) for each class and operations for the interfaces.
- Specify only the most required responsibilities. Do not clutter the classes and interfaces with unwanted information.

8.2-8.3 Check your progress

True or false

1. The purpose of a class diagram is to model the dynamic view of an application.
2. Class diagram is the only diagram which finds direct relation to object oriented languages.
3. Class diagram is very much useful in the construction of the application.

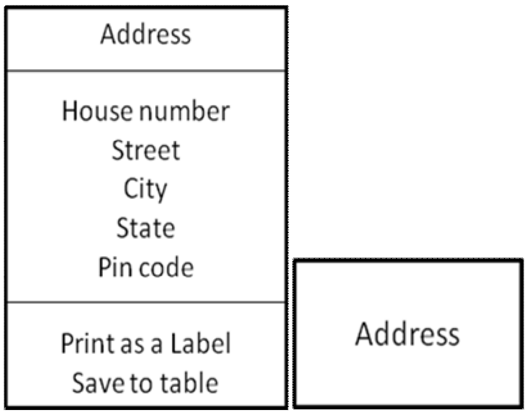
8.4 ELEMENTS OF CLASS DIAGRAM

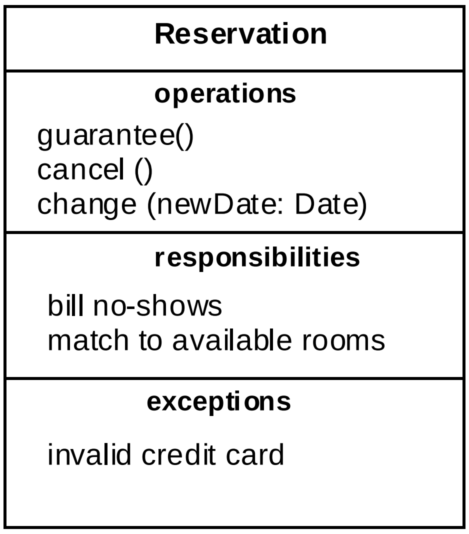
8.4.1 Class

A class is represented as a rectangle with three compartments. The top compartment is used to write the name of the class. The middle compartment is used for listing the attributes of that class and the last compartment is used to list the operations of that class.

Sometimes if there is no need to show all the attributes and operations simple write the class name in a rectangle which is not compartmentalized.

Sometimes more compartments can also be used. In this case the compartments are named and the corresponding responsibilities are written below the compartment name.



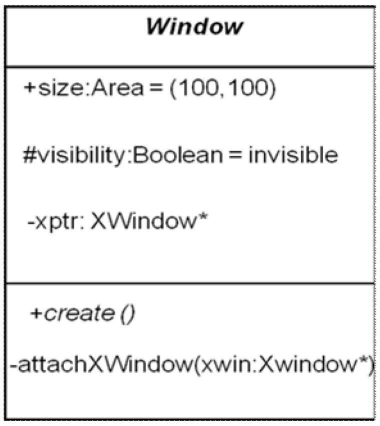


8.4.2 Visibility

There are three types of visibility modes available with the UML as with many other object oriented languages. These are public (seen by all), private (seen by itself) and protected (seen by the derived/child/subclass only). These modes can be used with the responsibilities of a class or roles in the associations.

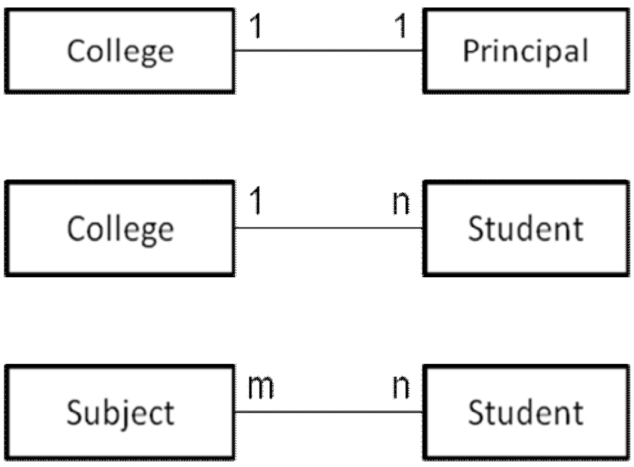
This is shown by the symbols + for public, - for private and # for protected. These symbols precede the name of the attribute, operation or role.

+size, -attach( ), #draw( ), +user, -password


Window
+size:Area = (100,100)
#visibility:Boolean = invisible
-xptr: XWindow*
+create ()
-attachXWindow(xwin:Xwindow*)

8.4.3 Multiplicity

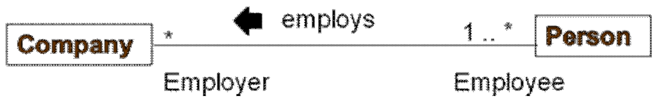
The classes involved in a relationship may have a relationship which is one-to-one, one-to-many or many-to-many. That means one instance of a class may be related to one instance of another class, one instance of a class may be related to many instances of another class, or many instances of a class may be related to many instances of another class. This is indicated by a number or n or m or *.



8.4.4 Association

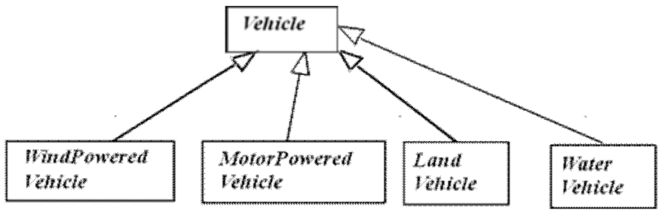
It is the simplest relationship that can exist between a pair of classes. The association can have multiplicity. This relationship can also be identified by a name. Also a direction of the relationship may be specified. The role of the participating classes also can be mentioned.

It is shown by a plain line joining two classes.



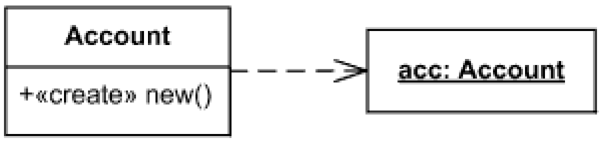
8.4.5 Generalization-specialization

This is the same kind of relationship which was explained under inheritance. This is the equivalent of the superclass-subclasses hierarchy.



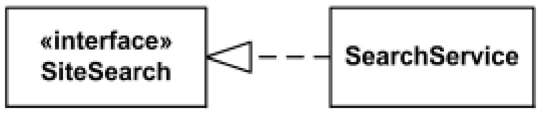
8.4.6 Dependence

When one class changes due to the change in another class, the relationship between these classes is shown by this relationship. This is shown by a dashed line with normal arrowhead.



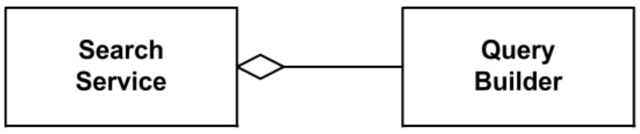
8.4.7 Realization

This is typically shown between a use case and the collaboration or an interface and the class or component implementing it.



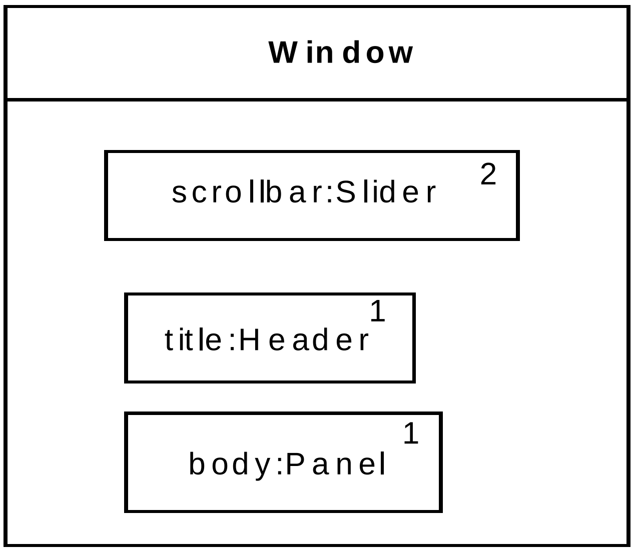
8.4.8 Aggregation

This is a whole-part relationship. Here the part could be part of more than one wholes. This is a weak relationship. When the whole is deleted the part may still exist. This is represented by a diamond at the end of an association link. The diamond is toward the whole end.



8.4.9 Composition

This is a whole-part relationship. Here the part is a part of exactly one whole. This is a strong relationship. When the whole is deleted the part also gets deleted. This is represented by a filled diamond at the end of an association link. The diamond is toward the whole end.



8.4 Check your progress

Fill in the blanks

1. Typically in a class representation, the middle compartment is used to write …………………….
2. There are ………………..types of visibility modes available.
3. The classes can be involved in ……………….., ……………………or ……………………relationships.
4. …………………………., ...represents inheritance.
5. When one class changes due to the change in another class, it is represented by ……………………….
6. In ……………………..the part could be part of more than one wholes whereas in ……………………..a part is a part of exactly one whole.

8.5 SUMMARY

The class diagrams give the static view of the system. This diagram shows different classes, their interactions to fulfill the requirements of the system. The classes and the relationships are the main elements of this diagram.

8.6 CHECK YOUR PROGRESS – *ANSWERS*

8.2-8.3

1. False
2. True
3. True

8.4

1. Attributes
2. Three
3. one-to-one, one-to-many, many-to-many
4. generalization, specialization
5. dependence
6. aggregation, composition

8.7 QUESTIONS FOR SELF - STUDY

1. What are classes and how are they formed?
2. Design a class ‘BOOK’.
3. Name the three levels of visibility.
4. How is multiplicity represented in relationships?
5. What is the difference between a composition and an aggregation?
6. Can a class diagram show the dynamic behavior of a system?

8.8 SUGGESTED READINGS

1. *Unified Modeling Language User Guide by* Grady Booch,James Rumbaugh, Ivar Jacobson
2. *UML 2 For Dummies* by Michael Jesse Chonoles, James A. Schardt

***

NOTES

NOTES