This diagram shows that each *Person* may be a member of zero or more *Departments* and that each *Department* must have at least one *Person* as a member. This diagram goes on to indicate that each *Person* may be the manager of zero or more *Departments*. All of these semantics can be expressed using simple UML.

However, to assert that a manager must also be a member of the department is something that cuts across multiple associations and cannot be expressed using simple UML. To state this invariant, we have to write a constraint that shows the manager is a subset of the members of the Department, connecting the two associations and the constraint by a dependency from the subset to the superset.

40. List and explain four structural diagrams in UML.

Ans:

UML has designated four diagrams namely,

1. Class diagram – Class, interfaces, and collaborations
2. Object diagram – Objects
3. Component diagram – Components
4. Deployment diagram – Nodes

Class Diagram

A class diagram shows a set of classes, interfaces, and collaborations and their relationships. Class diagrams are the most common diagram found in modeling object-oriented systems. We use class diagrams to illustrate the static design view of a system. Class diagrams that include active classes are used to address the static process view of a system.

Object Diagram

An object diagram shows a set of objects and their relationships. We use object diagrams to illustrate data structures, the static snapshots of instances of the things found in class diagrams. Object diagrams address the static design view or static process view of a system just as do class diagrams, but from the perspective of real or prototypical cases.

Component Diagram

A component diagram shows a set of components and their relationships. We use component diagrams to illustrate the static implementation view of a system. Component diagrams are related to class diagrams in that a component typically maps to one or more classes, interfaces, or collaborations.

Deployment Diagram

A deployment diagram shows a set of nodes and their relationships. We use deployment diagrams to illustrate the static deployment view of an architecture. Deployment diagrams are related to component diagrams in that a node typically encloses one or more components.

41. What are the five behavioural diagrams of UML ?

Ans :

The UML's five behavioural diagrams are used to visualize, specify, construct, and document the dynamic aspects of a system.

The UML's behavioural diagrams are roughly organized around the major ways we can model the dynamics of a system.

1. Use case diagram	Organizes the behaviours of the system
2. Sequence diagram	Focused on the time ordering of messages
3. Collaboration diagram	Focused on the structural organization of objects that send and receive messages.
4. Statechart diagram	Focused on the changing state of a system driven by events.
5. Activity diagram	Focused on the flow of control from activity to activity.

42. Explain how do you model different levels of abstraction.

Ans :

Modeling Different Levels of Abstraction

There are two ways to model a system at different levels of abstraction :

1) by presenting diagrams with different levels of detail against the same model, or
2) by creating models at different levels of abstraction with diagrams that trace from one model to another.

The following are the steps to model a system at different levels of abstraction by presenting diagrams with different levels of detail.

1. Consider the needs of the readers, and start with a given model.
2. If the reader is using the model to construct an implementation, he'll need diagrams that are at a lower level of abstraction, which means that they'll need to reveal a lot of detail. If the reader is using the model to present a conceptual model to an end user, he'll need diagrams that are at a higher level of abstraction, which means that they'll hide a lot of detail.
3. Depending on where we land in this spectrum of low-to-high levels of abstraction, create a diagram at the right level of abstraction by hiding or revealing the following four categories of things from our model :
   i) *Building blocks and relationships :* Hide those that re not relevant to the intent of our diagram or the needs of our reader.

72. What is a class diagram ? What are the contents and uses of class diagrams?

Ans :

Class Diagrams

Class diagrams are the most common diagram found in modeling object oriented systems. A class diagram is a diagram that shows a set of classes, interfaces, and collaborations and their relationships. Graphically, a class diagram is a collection of vertices and arcs.

Class diagrams are important not only for visualizing, specifying and documenting structural models, but also for constructing executable systems through forward and reverse engineering.

Common Properties

A class diagram is just a special kind of diagram and shares the same common properties as do all other diagrams – a name and graphical content that are a projection into a mode.

Contents

Class diagrams commonly contain the following things :

- Classes
- Interfaces
- Collaborations
- Dependency, generalization and association relationship.

a) **Classes :** "Classes are the collection of objects of similar type". Classes plays a major role in class diagrams.

b) **Interfaces :** "Interfaces are collection of operations which specify a service to a class".

- Interfaces focuses on the externally visible behaviour of an element.
- The behaviour may be a complete or only a part of it.
- Therefore interface only specifies but does not implements.

c) **Collaborations :**

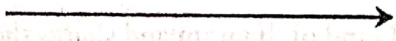
- Collaborations specifies a unit where different element work together to provide a cooperative behaviour.
- The collaborations are the most essential things to make up the systems.
- During implementation, collaborations can have structural as well as behavioral semantics.

d) **Relationships :** Whenever any diagram represents a set of things, relations bonds these things. Basically there are three types of relationships most comm used. They are :

- **Dependency :** Dependency is a type of relationship between the two enti with change caused to one entity may effect the other entity. These are genera represented by a dashed line.

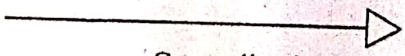
Dependency

- **Association :** It represents a structural relationship, connecting different objects This is generally represented by a solid line.



Association

- **Generalization :** Generalization is termed as a "specialized relationship". Here object of one entity can be substituted with the objects of another entity. The entity whose objects are substituted as known as a parent entity and the entity which is providing objects for replacement is known as child entity. This is generally represented by a solid line with a diamond head.



Generalization

Common Uses of Class Diagrams

1. Class diagrams are very essential to model entire details of a system.
2. It models different collaborations. These collaborations specifies a unit where different elements work together to provide a corporate behaviour.
3. Class diagrams also provide several mechanisms to model logical database. Here database can be referred to as a place where large amounts of data can be stored. Modelling such database requires many complex properties which are provided by class diagrams.

In UML, we use class diagrams to visualize the static aspects of these building blocks and their relationships and to specify their details for construction, as we can see in figure below.

73. Enumerate the steps to model simple collaborations.

Ans :

When we create a class diagram, we just model a part of the things and relationships that make up our system's design view. For this reason, each class diagram should focus on one collaborations at a time.

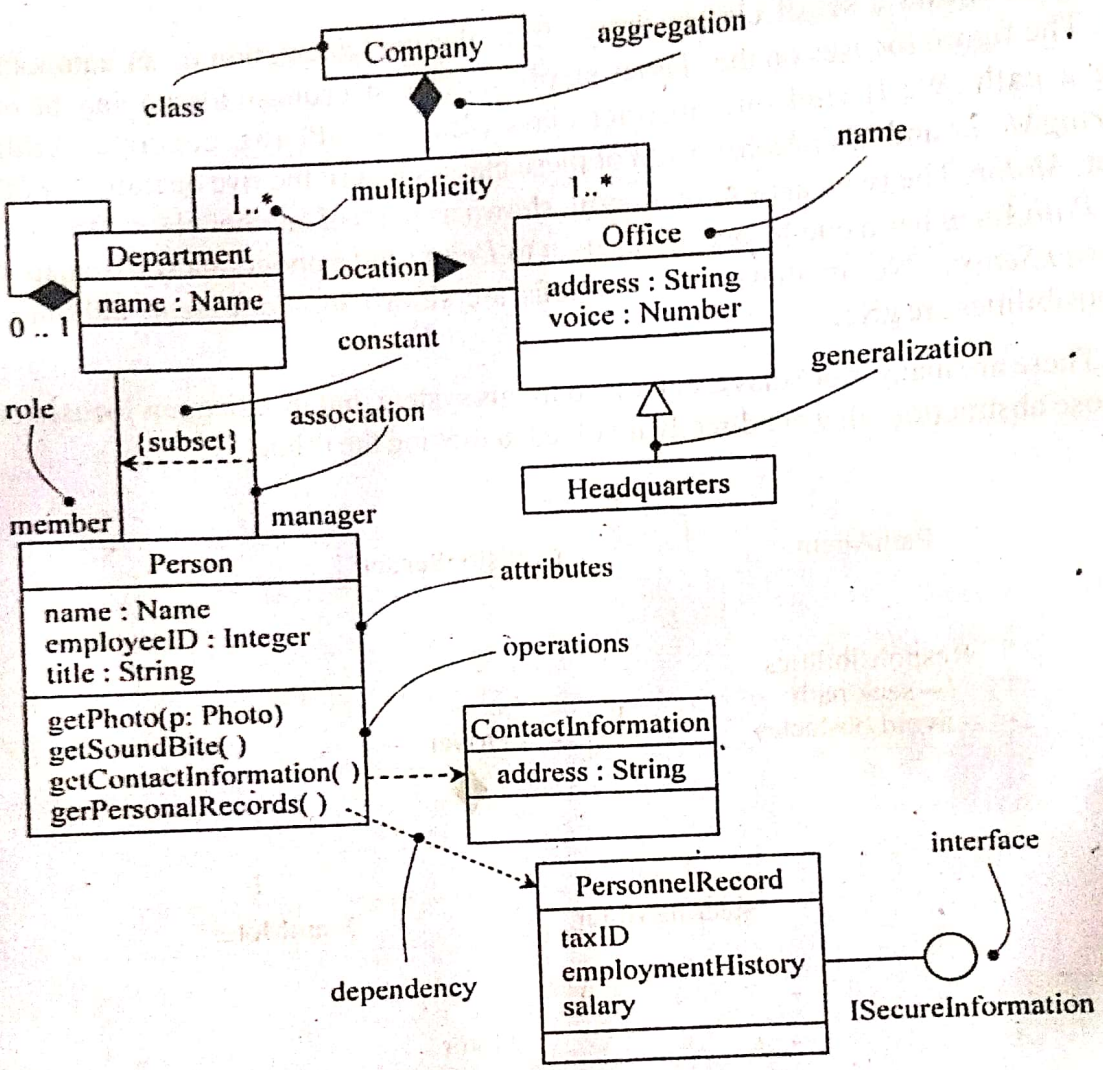


Fig. : Class Diagram

Steps to model a simple collaboration :

1) Identify the mechanism we would like to model. A mechanism represents some function or behaviour of the part of the system we are modeling that results from the interaction of a society of classes, interfaces, and other things.

2) Identify the classes, interfaces, and other collaborations that participate in this collaboration. Identify the relationships among these things, we well.

3) Use scenarios to walk through these things. By this we discover parts of our model that were missing and parts that were just plain semantically wrong.

4) Be sure to populate these elements with their contents. For classes, start with getting a good balance of responsibilities. Then, over time, turn these into concrete attributes and operations.

Figure shows a set of classes drawn from the implementation of an autonomous robot. The figure focuses on the classes involved in the mechanism for moving the robot along a path. We'll find one abstract class (*Motor*) with two concrete children, *StreeringMotor* and *MainMotor*. Both of these classes inherit the five operations of their parent, *Motor*. The two classes are, in turn, shown as parts of another class, *Driver*. The class *PathAgent* has a one-to-one association to *Driver* and a one-to-many association to *CollisionSensor*. No attributes or operations are shown for *PathAgent*, although its responsibilities are given.

There are many more classes involved in this system, but this diagram focuses only on those abstractions that are directly involved in moving the robot.

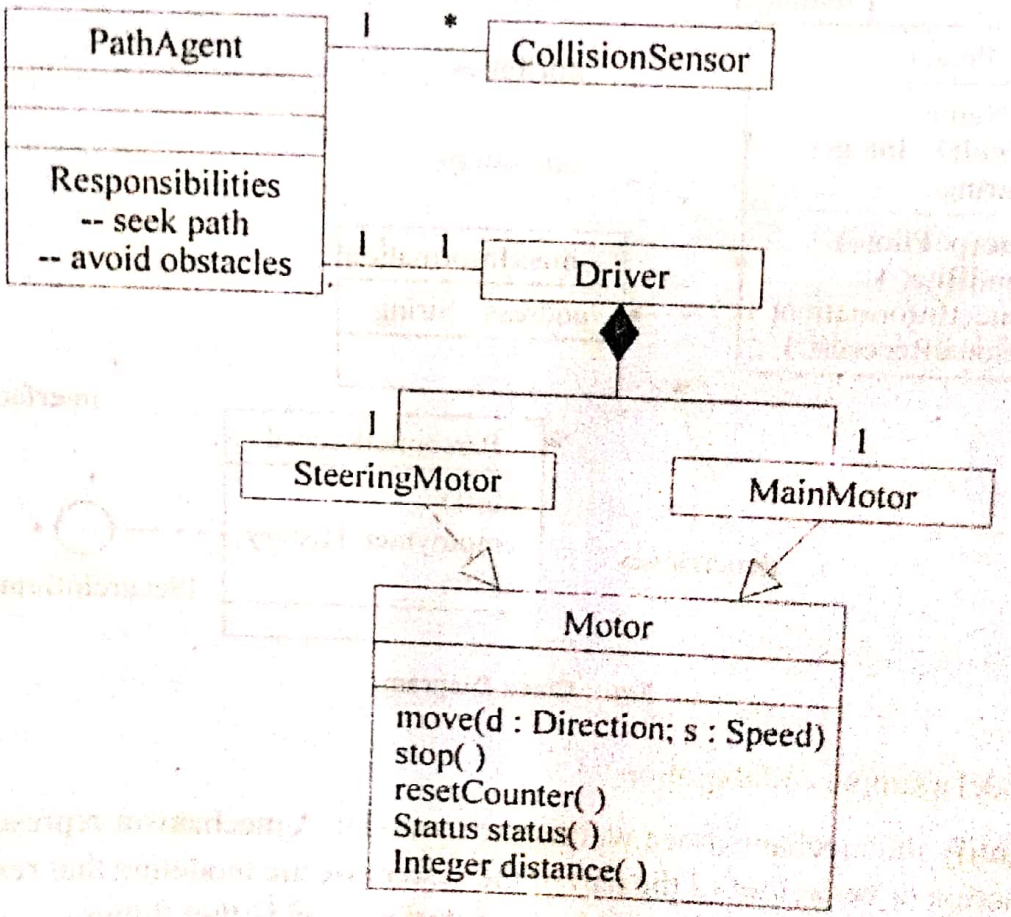


Fig. : Modeling Simple Collaborations

74. What are the steps involved in modeling a logical database schema ?

Ans :

Modeling Logical Database Schema

UML is a language, used for modelling different kinds of systems. The main features of UML will be demonstrated especially when it used in modelling an object-oriented database or relational database or hybrid object/relation database. These systems posses many profound objects, which can be stored/retrieved depending on the requirements.

Here are the following steps, which are essential in modelling logical database schema.

Step 1 : Select the classes from the existing model. The selection should be made basing on a condition that this "position should exceed the lifetime of their application".

Step 2 : Evaluate a class diagram and insert the classes (which was selected in step 1) in it if required, also market them using tagged values.

Step 3 : Make these classes more elucidating by enlarging their representations.

Step 4 : Reduce redundancy if require apply intermediate abstractions, which is essential tool in simplifying complexity.

Step 5 : To make these classes more elucidating, stress even on their behaviour. This can be achieved by elaborating various operations which reflects data integrating and data access.

Step 6 : Basing on the requirements, apply tools which reverts logical to physical designs respectively.

Hence the steps mentioned above forms the essentials, while dealing with modelling the logical database schema.

An example class diagram :

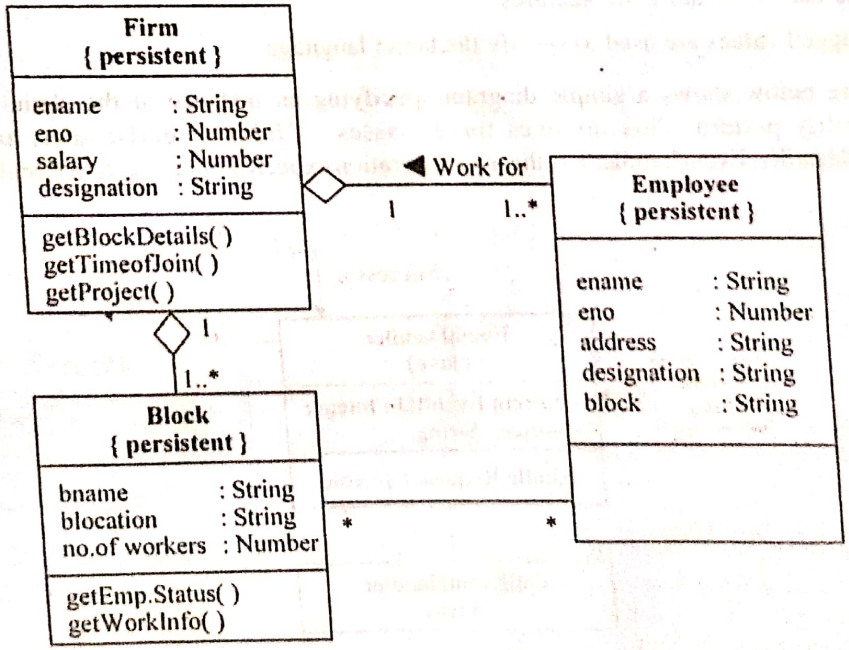


Fig. : Modelling a Schema

76. What is an object diagram ? What are the common properties, contents and uses of object diagrams ?

Ans :

Object Diagrams

An object diagram is a diagram that shows a set of objects and their relationships at a point in time. Graphically, an object diagram is a collection of vertices and arcs.

Common Properties

An object diagram is a special kind of diagram and shares the same common properties as all other diagrams – that is, a name and graphical contents that are a projection into a model.

Contents

Object diagrams commonly contain

1) Objects
2) Links

Object diagrams may contain notes and constraints. Object diagrams may also contain packages or subsystems, both of which are used to group elements of our model into larger chunks.

An object diagram is essentially an instance of a class diagram or the static part of an interaction diagram. In either case, an object diagram contains primarily objects and links, and focuses on concrete or prototypical instances. Both component diagrams and deployment diagrams may contain instances, and if they contain only instances (and no messages), they too are considered to be special kinds of object diagrams.

Object diagrams are used to model the static design view or static process view of a system just as we do with class diagrams but from the perspective of real or prototypical instances. This view primarily supports the functional requirements of a system – that is, the services the system should provide to its end users. Object diagrams let us to model static data structures.

An object diagram covers a set of instances of the things found in a class diagram. An object diagram, therefore, expresses the static part of an interaction, consisting of the objects that collaborate, but without any of the messages passed among them. In both cases, an object diagram freezes a moment in time, as in figure below.

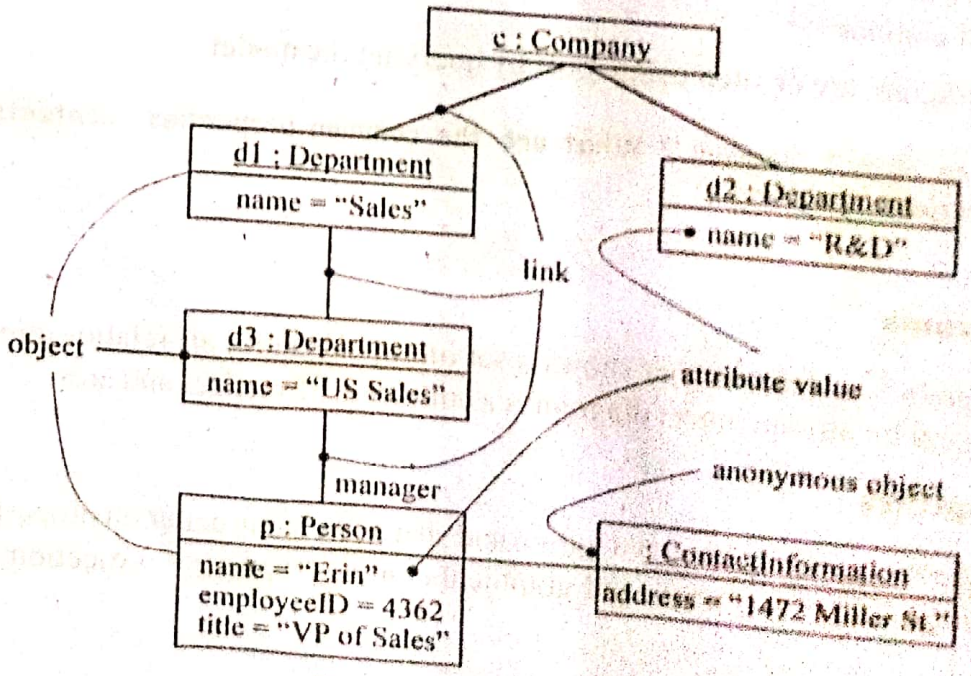


Fig.: An Object Diagram

77. With an aid of example explain how to model object structures.

Ans :

The following are the steps to model an object structure :

Step 1 : Identify the mechanism we would like to model. A mechanism represents some function or behaviour of the part of the system we are modeling that results from the interaction of a society of classes, interfaces, and other things.

Step 2 : For each mechanism, identify the classes, interfaces, and other elements that participate in this collaboration; identify the relationships among these things, as well.

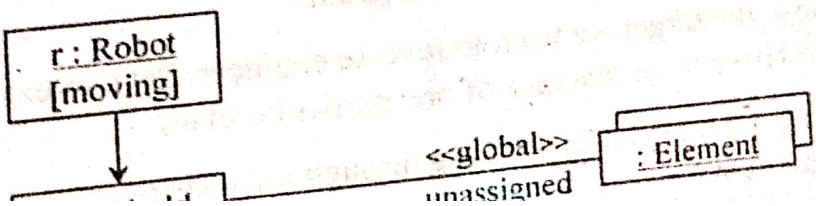
Step 3 : Consider one scenario that walks through this mechanism. Freeze that scenarios at a moment in time, and render each object that participates in the mechanism.

Step 4 : Expose the state and attribute values of each such object, as necessary, to understand the scenario.

Step 5 : Similarly, expose the links among these objects, representing instances of associations among them.

Example :

Figure below shows a set of objects drawn from the implementation of an autonomous robot. This figure focuses on some of the objects involved in the mechanism used by the robot to calculate a model of the world in which it moves. There are many more objects involved in a running system, but this diagram focuses on only those abstractions that are directly involved in creating this world view.



This figure indicates, one object represents the robot itself (*r*, an instance of *Robot*) and *r* is currently in the state marked *moving*. This object has a link to *w*, an instance of *World*, which represents an abstraction of the robot's world model. This object has a link to a multiobject that consists of instances of *Element*, which represent entities that the robot has identified but not yet assigned in its world view.

W is linked to two instances of *Area*. One of them (*a2*) is shown with its own links to three *Wall* and one *Door* object. Each of these walls is marked with its current width, and each is shown linked to its neighboring walls. As this object diagram suggests, the robot has recognized this enclosed area, which has walls on three sides and a door on the fourth.

78. Explain about forward and reverse engineering object diagrams.

Ans :

Forward and Reverse Engineering

Forward engineering (the creation of code from a model) an object diagram is theoretically possible but pragmatically of limited value. In an object-oriented system, instances are things that are created and destroyed by the application during run time.

Reverse engineering (the creation of a model from code) an object diagram is a very different thing. In fact, while we are debugging our system, this is something that we or our tools will do all the time.

For example, if we are chasing down a dangling link, we'll want to literally or mentally drawn an object diagram of the affected objects to see where, at a given moment in time, an object's state or its relationship to other objects is broken.

The steps to reverse engineer an object diagram.

Step 1 : Chose the target we want to reverse engineer. Set context inside an operation or relative to an instance of one particular class.

Step 2 : Using a too or simply walking through a scenarios, stop execution at a certain moment in time.

Step 3 : Identify the set of interesting objects that collaborate in that context and render them in an object diagram.

Step 4 : As necessary to understand their semantics, expose these object's states.

Step 5 : Identify the links that exist among these objects.

Step 6 : If the diagram ends up overly complicated, prune it by eliminating objects that are not important to the questions about the scenario we need answered. If our diagram is too simplistic, expand the neighbors of certain interesting objects and expose each object's state more deeply.

80. Draw a class diagram for a school information system.

Ans :

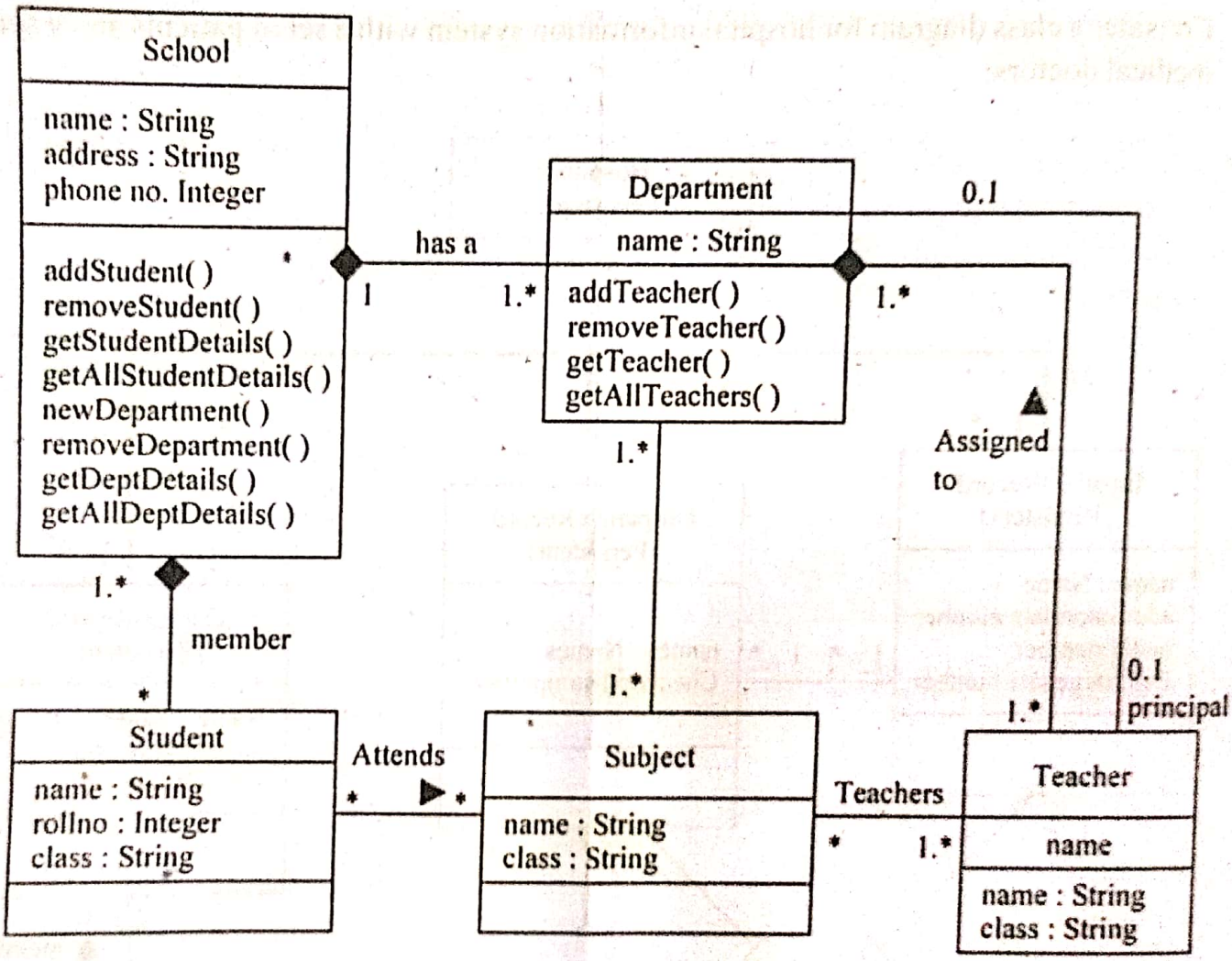


Fig. : Class Diagram for a School Information System

-:-:-:-

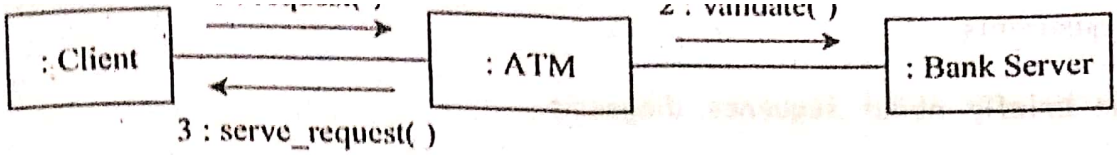


Fig. : Flat Sequence

These flat sequences are usually used to model the interactions where there exists a simple progression of control from one step to another.

5. What are interaction diagram ? What are their contents and common properties ?

Ans :

Interaction Diagrams

An interaction diagram shows an interaction, consisting of a set of objects and their relationships, including the messages that may be dispatched among them.

The two types of interaction diagrams are :

1. Sequence diagrams
2. Collaboration diagrams

1. **Sequence Diagrams :** 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.
2. **Collaboration Diagrams :** A collaboration diagram is an interaction diagram that emphasizes the structural organisation of the objects that send and receive messages. Graphically, a collaboration diagram is a collection of vertices and arcs.

Common Properties

An interaction 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.

Contents

Interaction diagrams commonly contain :

- Objects
- Links
- Messages

A sequence diagram emphasizes the time ordering of messages. We typically form a sequence diagram by first placing the objects that participate in the interaction at the top of our diagram, across the X axis.

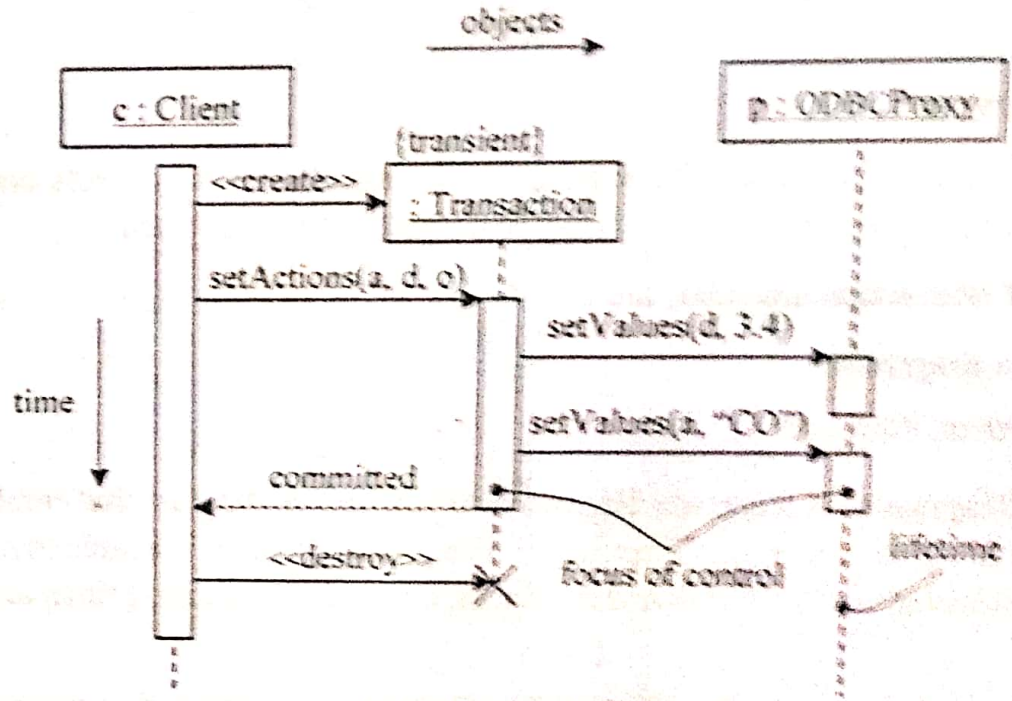


Fig. : Sequence Diagram

Typically, we place the objects that initiates the interaction at the left, and increasingly more subordinate objects to the right. Next, we 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.

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

1. The object life line and
2. The focus of control

An object lifeline is the vertical dashed line that represents the existence of an object over a period of time. Most objects that appear in an interaction diagram will be in existence for the duration of the interaction, so these objects are all aligned at the top of the diagram, with their lifelines drawn from the top of the diagram to the bottom. Objects may be

created during the interaction. Their lifelines start with the receipt of the message stereotyped as *create*. Objects may be destroyed during the interaction. Their lifelines end with the receipt of the message stereotyped as *destroy* (and are given the visual cue of a large *X*, marketing the end of their lives).

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. The top of the rectangle is aligned with the start of the action; the bottom is aligned with its completion (and can be marked by a return message). We can show the nesting of a focus of control (caused by recursion, a call to a self-operation, or by a call-back from another object) by stacking another focus of control slightly to the right of its parent (and can do so to an arbitrary depth).

7. Write about collaborations diagrams.

Ans :

Collaborations Diagrams

A collaboration diagram emphasizes the organisation of the objects that participate in an interaction. As Figure shows, we form a collaboration diagram by first placing the objects that participate in the interaction as the vertices in a graph. Next, we render the links that connect these objects as the arcs of this graph. Finally, we adorn these links with the messages that objects send and receive. This gives the reader a clear visual cue to the flow of control in the context of the structural organisation of objects that collaborate.

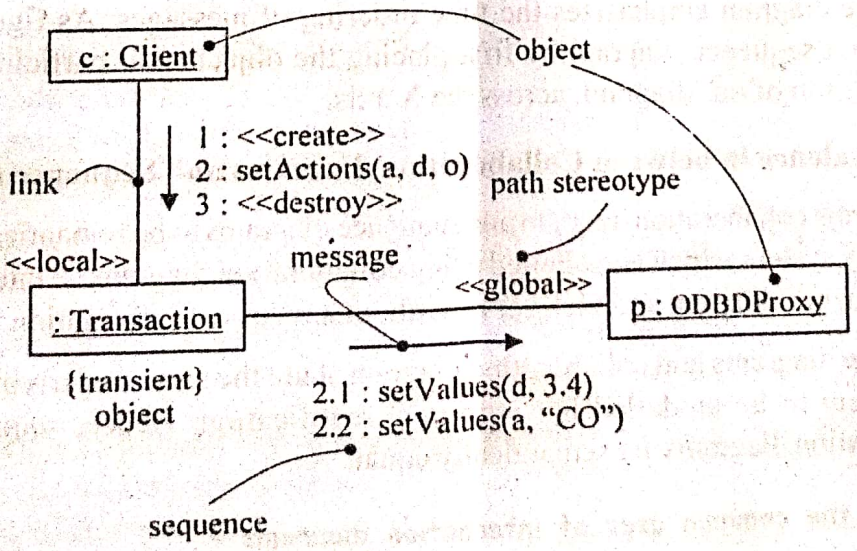


Fig. : Collaboration Diagram

Collaboration diagrams have two features that distinguish them from sequence diagrams.

1. The path
2. The sequence number

To indicate how one object is linked to another, we can attach a path stereotype to the far end of a link (such as *<<local>>*, indicating that the designated object is local to the sender). Typically, we will only need to render the path of the link explicitly for *local*, *parameter*, *global*, and *self* (but not *association*) paths.

To indicate the time order of a message, we prefix the message with a number (starting with the message numbered 1), increasing monotonically for each new message in the flow of control (2, 3 and so on). To show nesting, we use Dewey decimal numbering (1 is the first message; 1.1 is the first message nested in message 1; 1.2 is the second message nested in message 1; and so on). We can show nesting to an arbitrary depth. Along the same link, we can show messages, and each will have a unique sequence number.

8. What is semantic equivalence between sequence and colldboration diagrams?

Ans :

Collaboration diagrams organizes various objects participating in an interaction orderly. These objects can interact with each other using various messages. Thus, in short we can say that collaboration diagrams consists of objects, links and messages. The names of the objects are instance of classes. Sometimes, the name of an object also refers to the instance of components, nodes, etc. to which they are related.

The most important feature of the collaboration diagram is that, it is used to model dynamic view of the system.

A sequence diagram emphasizes the time ordering of messages. As figure below shows, we form a sequence diagram by first placing the objects that participate in the interaction at the top of our diagram, across the X axis.

Semantic equivalence in between Collaboration diagrams and Sequence diagrams

- UML terms collaboration diagram and sequence diagrams to be semantically equal, (i.e.,) any system which is modelled using collaboration diagram be interchanged into sequence diagram and vice versa, without any loss of information.
- Sequence diagrams and collaboration diagram share the same underlying logic of the system to be modelled but with new rectification. Hence, sequence and collaboration diagrams are semantically equal.

9. What are the common uses of interaction diagrams ?

Ans :

We use interaction diagrams to model the dynamic aspects of a system. These dynamic aspects may involve the interaction of any kind of instance in any view of a system's architecture, including instances of classes (including active classes), interfaces, components, and nodes.

When we use an interaction diagram to model some dynamic aspect of a system, we do so in the context of the system as a whole, a subsystem, an operation, or a class. We can also attach interaction diagrams to use cases (to model a scenario) and to collaborations (to model the dynamic aspects of a society of objects).

When we model the dynamic aspects of a system, we typically use interaction diagrams in two ways.

1) To model flows of control by time ordering.

2) To model flows of control by organisation.

Modeling a flow of control by organisation emphasizes the structural relationships among the instances in the interaction, along which messages may be passed. Collaboration diagrams do a better job of visualizing complex iteration and branching and of visualising multiple concurrent flows of control than do sequence diagrams.

10. Differentiate sequence and collaboration diagrams.

Ans :

Even though both sequence and collaboration diagrams show interaction among objects, there are some differences among them. A sequence diagram emphasizes the time ordering of messages whereas a collaboration diagram emphasizes the organisation of objects that participate in an interaction.

A sequence diagram has two features that distinguish them from collaboration diagrams.

1. **Object Lifeline :** Every sequence diagram carries an object lifeline. It represents the duration during which the object exists. The lifeline starts with the receipt of the message which is stereotyped as <<create>>. Objects may also be destroyed by a message stereotyped as <<destroy>>.
2. **Focus of Control :** It is represented by a long rectangle which shows the period of time during which an object performs some action.

A collaboration diagram has two features that distinguish them from sequence diagram.

1. **Path :** To indicate how one object interacts with other. We associate a path stereo to the end of link. Some of the path stereotypes are <<local>>, <<global>>, <<parameter>> etc.
2. **Sequence Number :** The messages in collaboration diagram are prefixed with sequence numbers to indicate time order of messages. Some UML tools provide sequence numbers for sequence diagrams also.

Both the sequence and collaboration diagrams are semantically equal. They are inter

convertible. Usually their interaction diagrams are used to model the flows of control by time ordering and to model flows of control by organisation. A single sequence diagram can describe only one scenario or situation. Thus is its disadvantage.

11. What are the steps to model the flow of control using interactions ?

Ans :

Modeling a Flow of Control

The most common purpose for which we'll use interactions is to model the flow of control that characterizes the behaviour of a system as a whole, including use cases, patterns, mechanisms, and frameworks, or the behaviour of a class or an individual operation.

When we model an interaction, we essentially build a storyboard of the actions that take place among a set of objects.

Following are the steps to model a flow of control :

Step 1 : Set the context for the interaction, whether it is the system as a whole, a class, or an individual operation.

Step 2 : Set the stage for the interaction by identifying which objects play a role; set their initial properties, including their attribute values, state, and role.

Step 3 : If the model emphasizes the structural organisation of these objects, identify the links that connect them, relevant to the paths of communication that take place in this interaction. Specify the nature of the links using the UML's standard stereotypes and constraints, as necessary.

Step 4 : In time order, specify the messages that pass from object to object. As necessary, distinguish the different kinds of messages; include parameters and return values to convey the necessary detail of this interaction.

Step 5 : Also to convey the necessary detail of this interaction, adorn each object at every moment in time with its state and role.

Example

Figure below shows a set of objects that interact in the context of a publish and subscribe mechanism (an instance of the observer design pattern). This figure includes three objects; *p* (a *StockQuotePublisher*), *s1*, and *s2* (both instances of *StockQuoteSubscriber*). This figure is an example of a sequence diagram, which emphasizes the time order of messages.

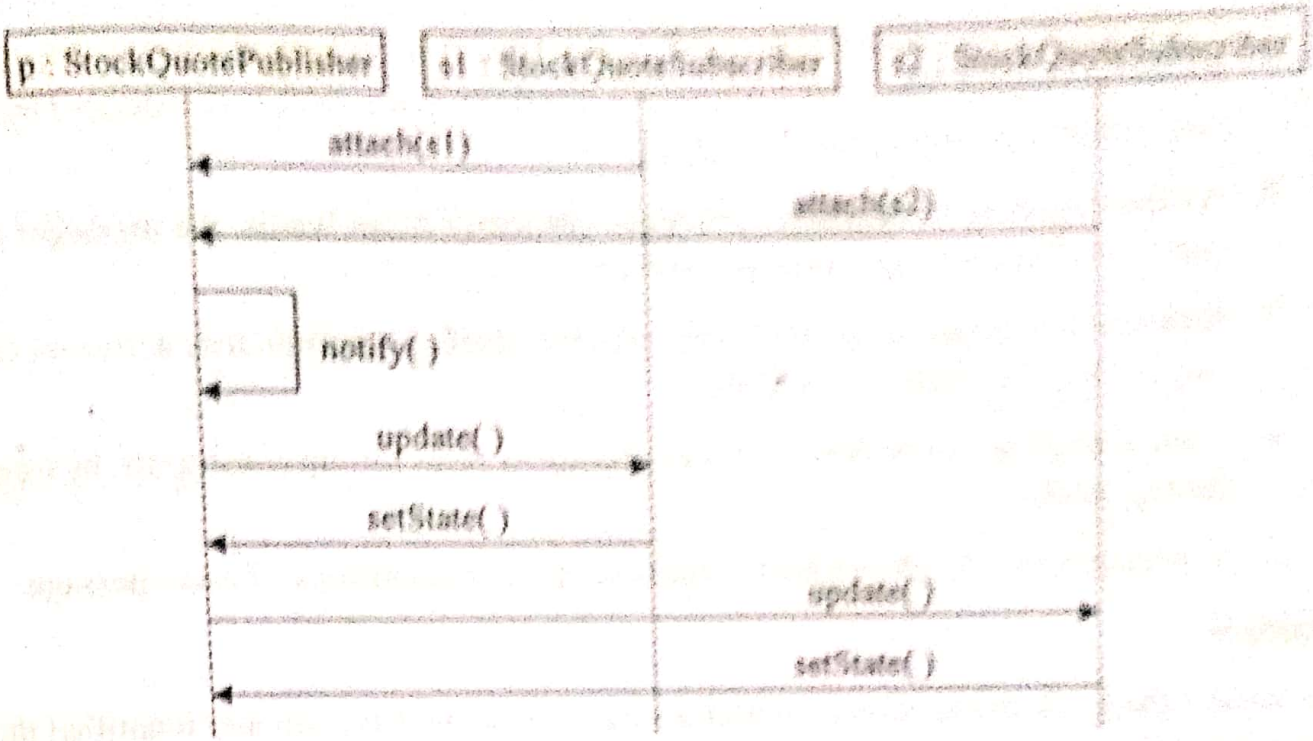


Fig. : Flow of Control by Time

Figure below is semantically equivalent to the previous one, but it is drawn as a collaboration diagram, which emphasizes the structural organization of the objects. This figure shows the same flow of control, but it also provides a visualization of the links among these objects.

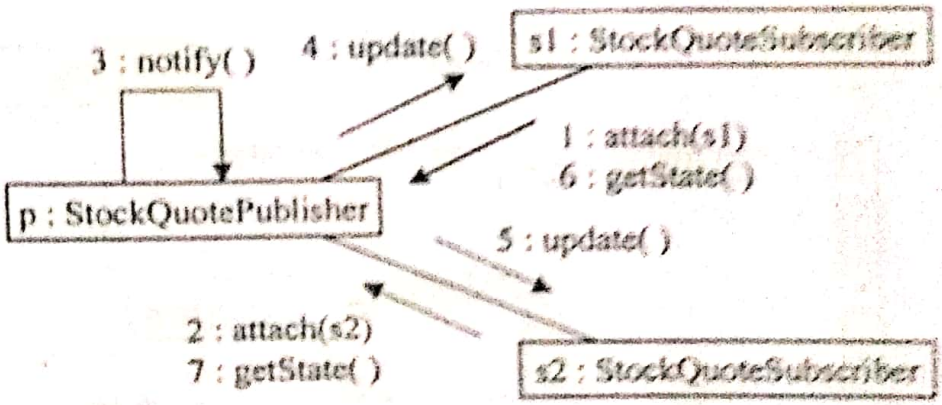


Fig. : Flow of Control by Organisation

12. List out various steps to model flows of control by time ordering.

Ans :

The various steps involved to model flows of control by time ordering are as follows :

1. Initially, we have to set the perspective or scope through which we can visualize the system or its parts.
2. Secondly, we need to consider the participants of the system (i.e., object roles). These objects are then illustrated in the sequence diagram.

3. Each object has a time in which it is active, that time period which is its life time, has to be created. The activation and termination of the objects is indicated by the corresponding stereotyped messages.
4. As the sequence of messages proceeds vertically down wards, the messages are depicted along with semantic properties.
5. Whenever message loops are required or the object is continuously active we can use "focus of control" to indicate it.
6. Time stamping can be done if it is necessary to show timing constraints, by using timing marks.
7. If required we can shows post-conditions and preconditions of each message.

Example :

If consider the bank transaction of withdrawing money, we have already identified the scope. The next step is to consider the objects involved in the transaction i.e., the customer, bank clerk and the teller. The bank clerk creates a transaction using stereotype <<creates>>. The respective active state is depicted on the life time through vertical rectangles called focus of control. The figure shows the modelling of flows of control by time ordering.

Withdrawing Money Transaction

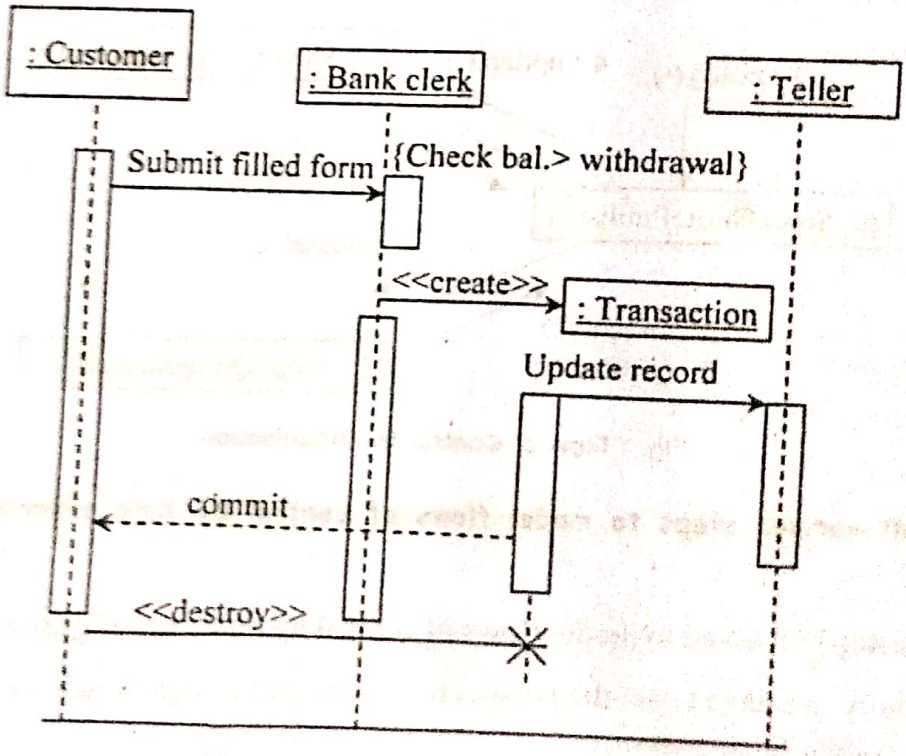


Fig. : Modelling of Flows of Control by Time Ordering

13. Enumerate the steps to model flows of control by organization.

Ans :

Below are the steps for modelling flow of control by organization :

Step 1 : Initially, determine the scenario where interaction takes place (*Eg.* It can be system on the whole or part of the system etc.)

Step 2 : As the scenario is known, now determine the objects participating in the interaction. Place the most essential object at the centre of the diagram surrounded by rest of the objects. Arrange them in a collaboration diagram.

Step 3 : Assign initial properties to each of these objects. It may happen that the values or stereotypes or status or roles of the objects may change significantly as the interaction proceeds. To narrate this consequence, use virtual objects and assign new properties to it (as per the requirement). Also, assign messages appended with suitable sequence numbers. Stereotypes "copy" and "become" can be used over new messages.

Step 4 : Now assign links to these objects such that the association links are added first followed by rest of the links. Here "global" and "local" stereotypes can be used.

Step 5 : Traverse to the initiating link (where the current interaction get triggered). Add appropriate message over this link along with suitable sequence number. It can happen that there may exists nesting of links. To manage such occasion use "Dewey decimal" numbering methodology.

Step 6 : If required to add time and scope over these messages, assign them accordingly.

Step 7 : If required add pre-conditions and post-conditions to these messages to increase its granularity.

Following figure depicting the modelling flows of control by organisation. Here the scenario is "purchasing of bus ticket".

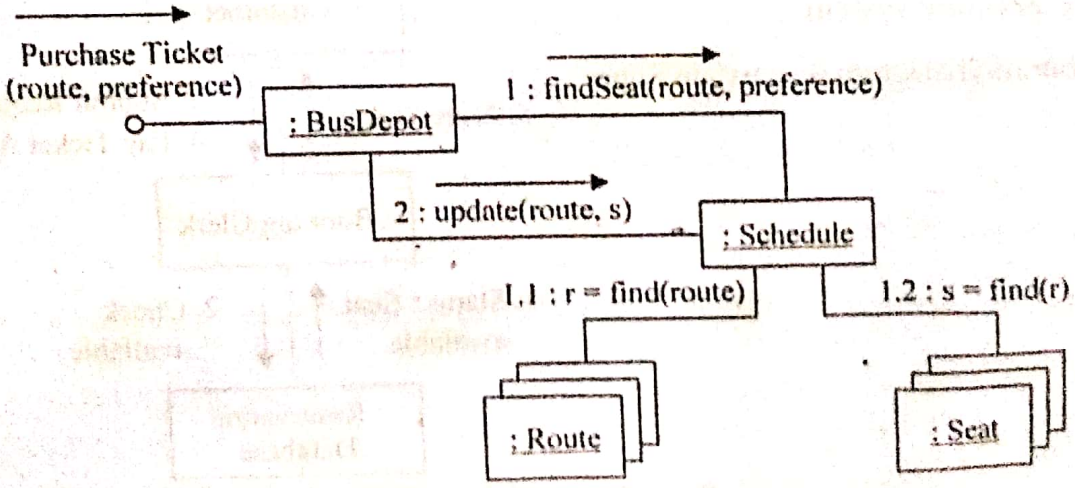


Fig. : Modelling the Flows of Control by Organisation

18. Define use case. How is it represented graphically in UML ?

Ans :

Use Cases

A use case is a description of a set of sequences of actions, including variants, that a system performs to yield an observable result of value to an actor.

A use case describes a set of sequences, in which each sequence represents the interaction of the things outside the system (its actors) with the system itself (and its key abstractions). These behaviours are in effect system-level functions that we use to visualize, specify, construct, and document the intended behaviour of the system during requirements capture and analysis.

A use case represents a functional requirement of the system as a whole. For example, one central use case of a bank is to process loans.

A use case involves the interaction of actors and the system. An actor represents a coherent set of roles that users of use cases play when interacting with these use cases. Actors can be human or they can be automated systems.

For example, in modeling a bank, processing a loan involves, among other things, the interaction between a customer and a loan officer.

We can apply use cases to the whole system. We can also apply use cases to part of the system, including subsystems and even individual classes and interfaces. Use cases applied to subsystems are excellent sources of regression tests; use cases applied to the whole system are excellent sources of integration and system tests.

The UML provides a graphical representation of a use case and an actor, as Figure below shows.

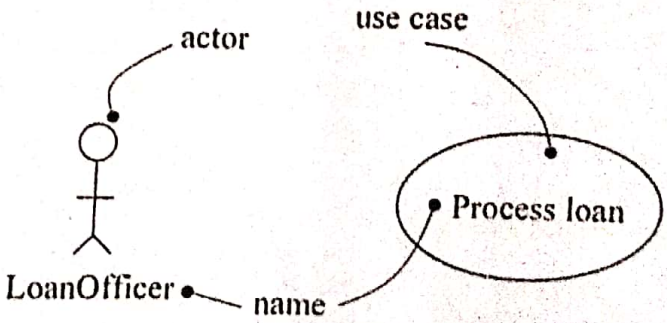


Fig. : Actors and Use Cases

19. Explain about use cases and actors.

Ans :

Use Cases and Actors

An actor represents a coherent set of roles that users of use cases play when interacting with these use cases. An actor represents a role that a human, hardware device, or even another system plays with a system.

For example, if we work for a bank, we might be a *LoanOfficer*. If we do our personal banking there, as well, we'll also play the role of *Customer*. An instance of an actor, therefore, represents an individual interacting with the system in a specific way.

Figure below indicates, actors are rendered as stick figures. We can define general kinds of actors (such as *Customer*) and specialize them (such as *CommercialCustomer*) using generalization relationships.

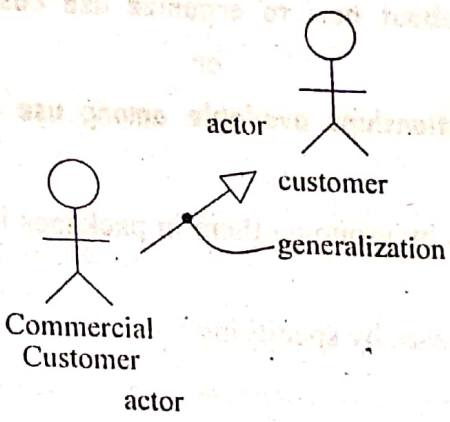


Fig. : Actors

Actors may be connected to use cases only by association. An association between an actor and a use case indicates that the actor and the use case communicate with one another, each one possibly sending and receiving messages.

20. What is the significance of use cases and collaborations ?

Ans :

A use case captures the intended behaviour of the system (or subsystem, class, or interface) we are developing, without having to specify how that behaviour is implemented. That's an important separation because the analysis of a system (which specifies behaviour) should, as much as possible, not be influenced by implementation issues (which specify how that behaviour is to be carried out).

However, we have to implement the use cases, and we do so by creating a society of classes and other elements that work together to implement the behaviour of this use case. This society of elements, including both is static and dynamic structure is modeled in the UML as a collaboration.

As figure below shows, we can explicitly specify the realization of a use case by a collaboration. A given use case is realized by exactly one collaboration, so we will not need to model this relationship explicitly.

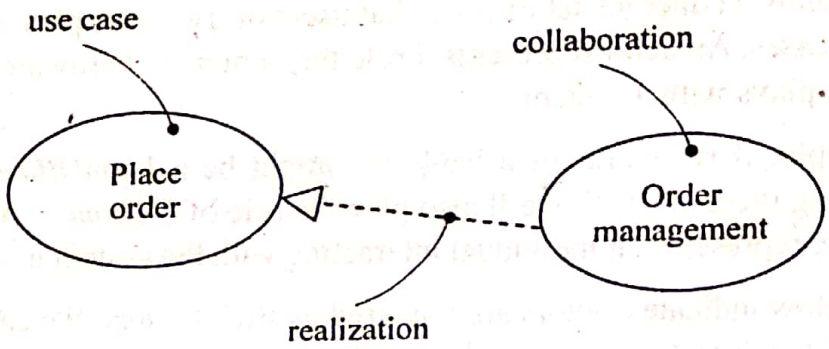


Fig. : Use Cases and Collaborations

21. Describe in detail about how to organize use cases.

or

Explain various relationships available among use cases.

Ans :

We can organize use cases by grouping them in packages in the same manner in which we can organize classes.

We can also organize use cases by specifying :

1) generalizations,
2) include, and
3) extend relationships among them.

We apply these relationships in order to factor common behaviour (by pulling such behaviour from other use cases that it includes) and in order to factor variants (by pushing such behaviour into other use cases that extent it).

Generalization

Whenever a given entity say entity1 inherits the behaviour of another entity easy entity2 then such consequence is referred as "inheritance". To express inheritance relationship in diagrams we use generalization relationship. It has to be noted that (in our case) entity1 completely inherits the behaviour of entity2, hence entity1 now then outs to be more powerful than entity2 since it inherits the behaviour of entity2 and also it has got it's own behaviour.

Example :

Following is an example diagram depicting generalization relationship among two levels of actors.

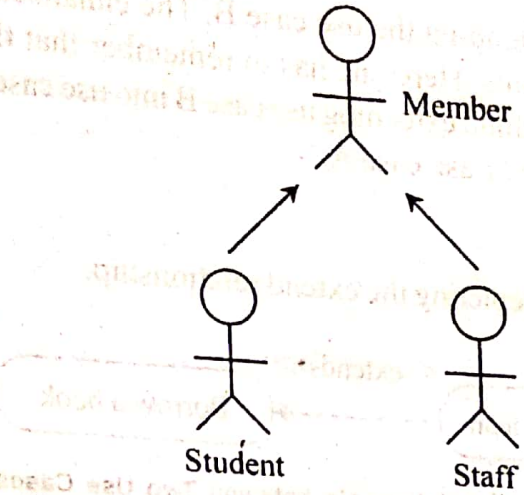


Fig. (a) : Generalization Relationship among Actors

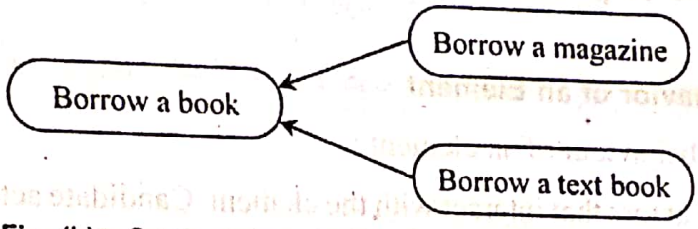


Fig. (b) : Generalization Relationship among Use Cases

Include Relationship

If "include" relationship is maintained between two use cases say use case A and use case B (where use case B being the base use case) it reflects that the behaviour of use case B is defined in use case A. Hence, use case B now possess to behaviours i.e., behaviours of use case B as well as it's own behaviour. The base use case i.e., use case A directly remains dependent on the external behaviour of use case B and the behaviour of use case B remains essential for use case A for sake of it's successful implementation.

Following is an UML notation depicting the include relationship.

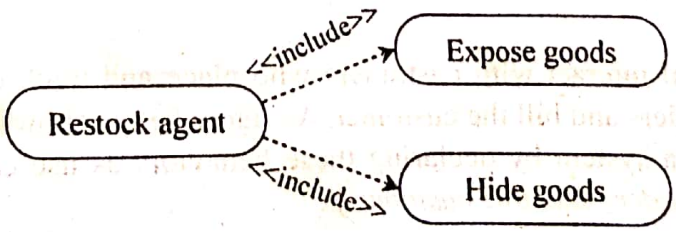


Fig. (c) : Include Relationship

Extend Relationship

The extend relationship when used between the two use cases say use cases A and use case B (where use case B remaining the base use case) it reflects that the functionality of

use case A got enhanced on extending the use case B. The enhanced feature of use case A can be observed at many points. Here one has to remember that the use case B exists independently also the reason behind extending use case B into use case A is to intentionally enhance the existing behaviour of use case A.

Example :

Following is an UML notation depicting the extend relationship.

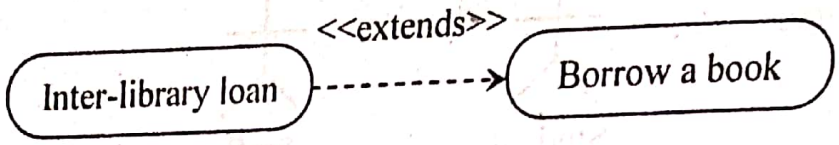


Fig. (d) : Extends Relationship between Two Use Cases

22. Enumerate the steps to model the behaviour of an element.

Ans :

Modeling the Behavior of an Element

Steps to model the behaviour of an element :

1. Identify the actors that interact with the element. Candidate actors include groups that require certain behaviour to perform their tasks or that are needed directly for indirectly to perform the element's functions.
2. Organize actors by identifying general and more specialized roles.
3. For each actor, consider the primary ways in which that actor interacts with the element. Consider also interactions that change the state of the element or its environment or that involve a response to some event.
4. Consider also the exceptional ways in which each actor interacts with the element.
5. Organize these behaviours as use cases, applying include and extend relationships to factor common behaviour and distinguish exceptional behaviour.

Example :

A retail system will interact with customers who place and track orders. In turn, the system will ship orders and bill the customer. As figure below shows, we can model the behaviour of such a system by declaring these behaviors as use cases (*Place order*, *Track order*, *Ship order*, and *Bill customer*).

Common behavior can be factored out (*Validate customer*) and variants (*Ship partial order*) can be distinguished, as well. For each of these use cases, we would include a specification of the behaviour, either by text, state machine, or interactions.

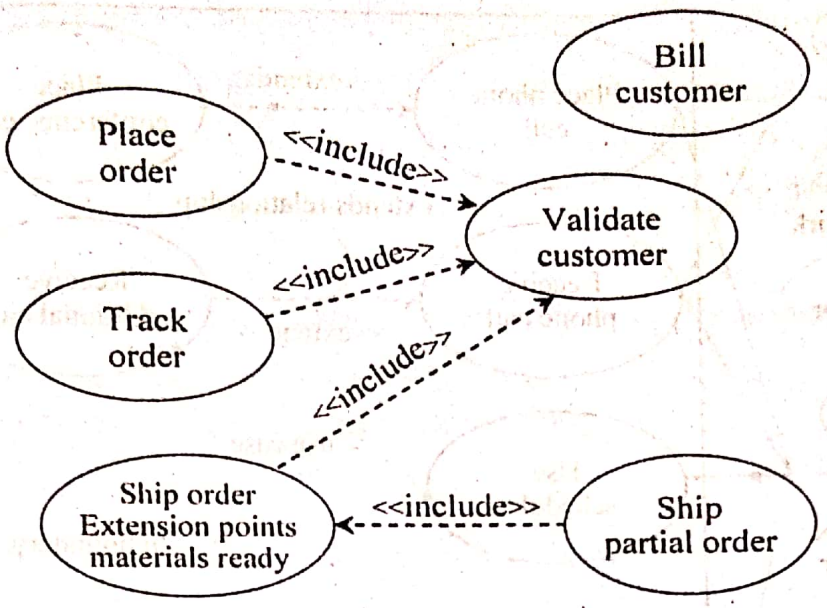


Fig. : Modeling the Behaviour of an Element

23. What is a use case diagram ? What are its common properties, contents and uses ?

Ans :

Use Case Diagram

Definition :

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

We apply use case diagrams to visualize the behaviour of a system, subsystem, or class so that users can comprehend how to use that element, and so that developers can implement that element. As figure below shows, we can provide a use case diagram to model the behaviour of that box – which most people would call a cellular phone.

Common Properties

Use case diagrams are well known to address static use case view of the system. It is their contents through which we can distinguish them from various other UML supported diagrams.

Contents

Use case diagrams commonly contain :

1) Use cases
2) Actors
3) Dependency, generalization, and association relationships

Like all other diagrams, use case diagrams may contain notes and constraints.

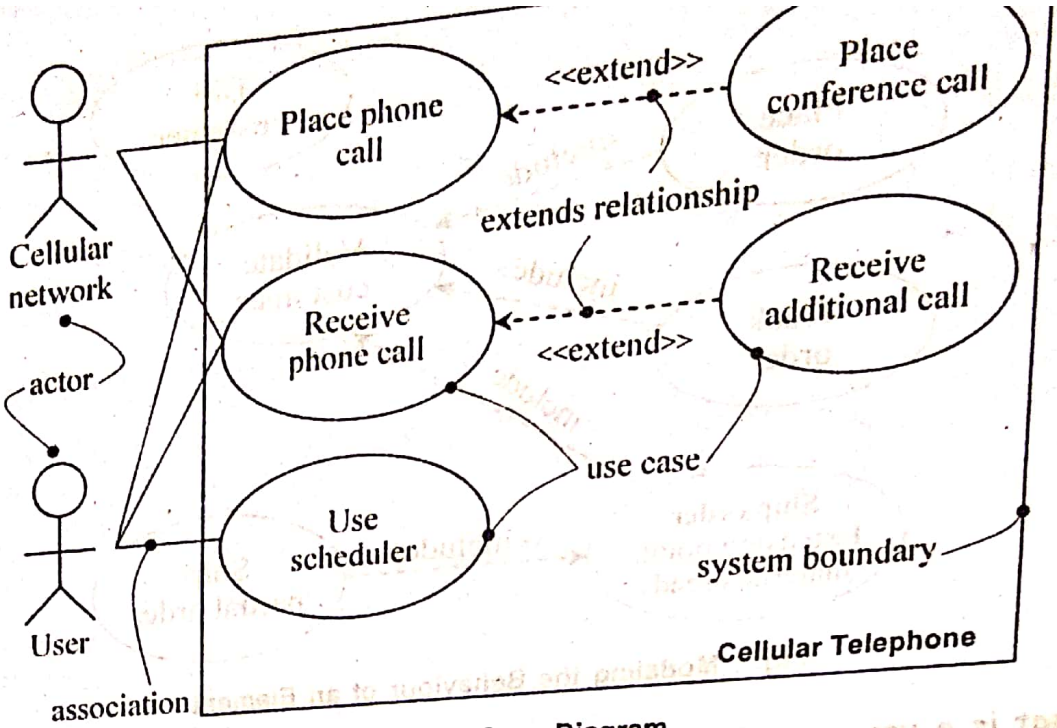


Fig. : A Use Case Diagram

Use case diagrams may also contain packages, which are used to group elements of our model into larger chunks. Occasionally, we'll want to place instances of use cases in your diagrams, as well, especially when we want to visualize a specific executing system.

Use

Use case diagrams are commonly used to visualize, specify and document the behaviour of an element contained in a system. Thus making systems subsystems and classes realistic by presenting them in the diagrams.

The various uses of use cases are as follows :

- The concepts of forward engineering supported by use case are used for testing the executable systems, also the reverse engineering concepts are applied in understanding the executable systems.
- They are widely used in modelling the context of a system. For doing this, they require a line to be drawn around the system by separating the actors to lie outside the system and also assist these actors to interact with the system.
- These diagrams are also recognized in modelling requirement of a system.

24. List out the various steps involved in modeling the context of a system.

Ans :

Following are the various steps to model the context of a system,

1. Identify the actors that surround the system by considering :
   i) Which groups require help from the system to perform their tasks.

ii) Which groups are needed to execute the system's functions.

iii) Which groups interact with external hardware or other software system.

iv) Which groups perform secondary functions for administration and maintenance.

2. Organize actors that are similar to one another in a generalization/specialization hierarchy.
3. Where it aids understandability, provide a stereotype for each such actor.
4. Populate a use case diagram with these actors and specify the paths of communication from each actor to the system's use cases.

Example :

Figure below shows the context of a credit card validation system, with an emphasis on the actors that surround the system.

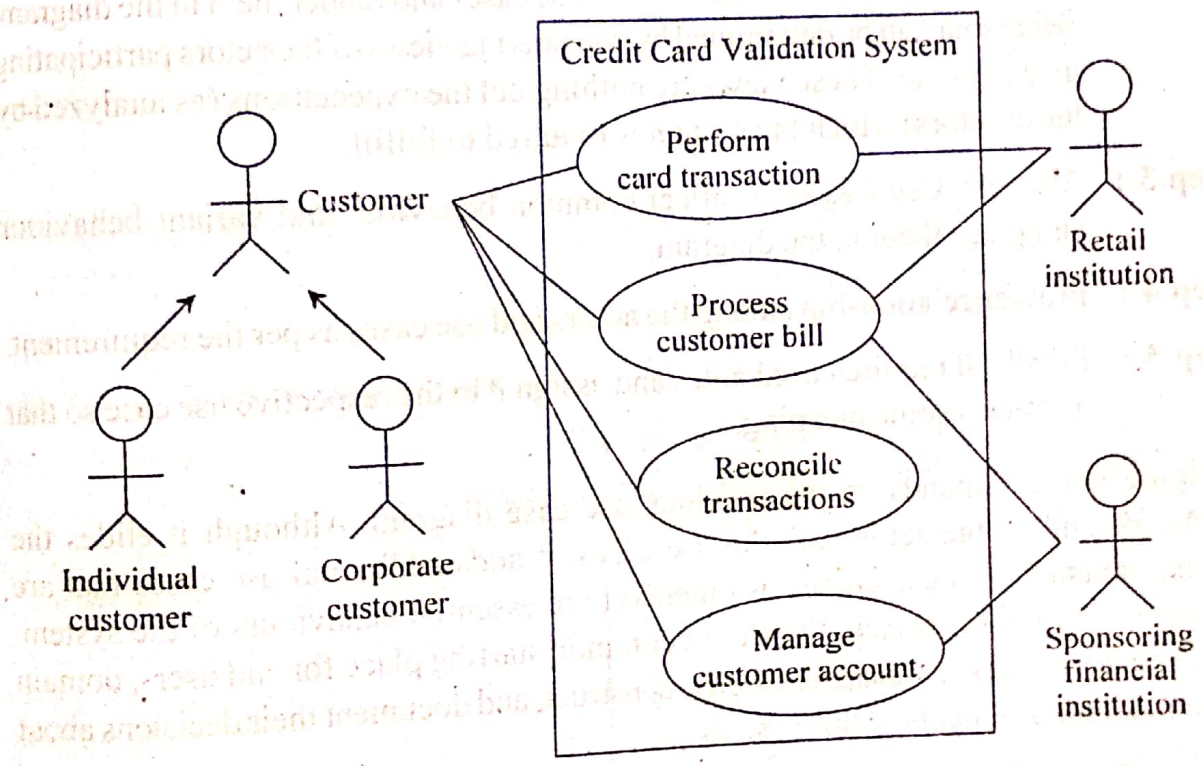


Fig. : Modeling the Context of a System

Customers, of which there are two kinds (*Individual customer* and *Corporate customer*). These actors are the roles that humans play when interacting with the system. In this context, there are also actors that represent other institutions, such as *Retail institution* (with which a *Customer* performs a card transaction to buy an item or a service) and *Sponsoring financial institution* (which serves as the clearing-house for the credit card account). In the real world, these latter two actors are likely software-intensive systems themselves.

25. Enumerate the steps to model the requirements of a system.

Ans :

Modeling the Requirements of a System

Requirements can be expressed in various forms, from unstructured text to expressions in a formal language, and everything in between. Most, if not all, of a system's functional requirements can be expressed as use cases, and the UML's use case diagrams are essential for managing these requirements.

Following are the steps to model the requirements of a system :

Step 1 : Initially determine the scene or the situation to modelled along with the actors remaining outside of the system.

Step 2 : As we are modelling the requirements of a system, these requirements can be in many forms say in the form of behaviour, feature or property of the system. Hence, adorn behaviours in use cases and render them in the diagram. Behaviour can be determined by considering views of the actors participating in the system. These views are nothing but the expectations (as analyzed by these actors) which the system is required to fulfill.

Step 3 : Also use Use Cases to reflect common behaviour and variant behaviour and place them in the diagram.

Step 4 : Provide relationship among the actors and use cases as per the requirement.

Step 5 : Finally, if required used notes and assign it to the respective use case so that it reflects some meaning.

Figure below expands on the previous use case diagram. Although it elides the relationships among the actors and the use cases, it adds additional use cases that are somewhat invisible to the average customer, yet are essential behaviours of the system. This diagram is valuable because it offers a common starting place for end users, domain experts, and developers to visualize, specify, construct, and document their decisions about the functional requirements of this system.

For example, *Detect card fraud* is a behaviour important to both the *Retail institution* and the *Sponsoring financial institution*. Similarly, *Report on account status* is another behaviour required of the system by the various institutions in its context.

The requirement modeled by the use case *Manage network outage* is a bit different from all the others because it represents a secondary behaviour of the system necessary for its reliable and continuous operation.

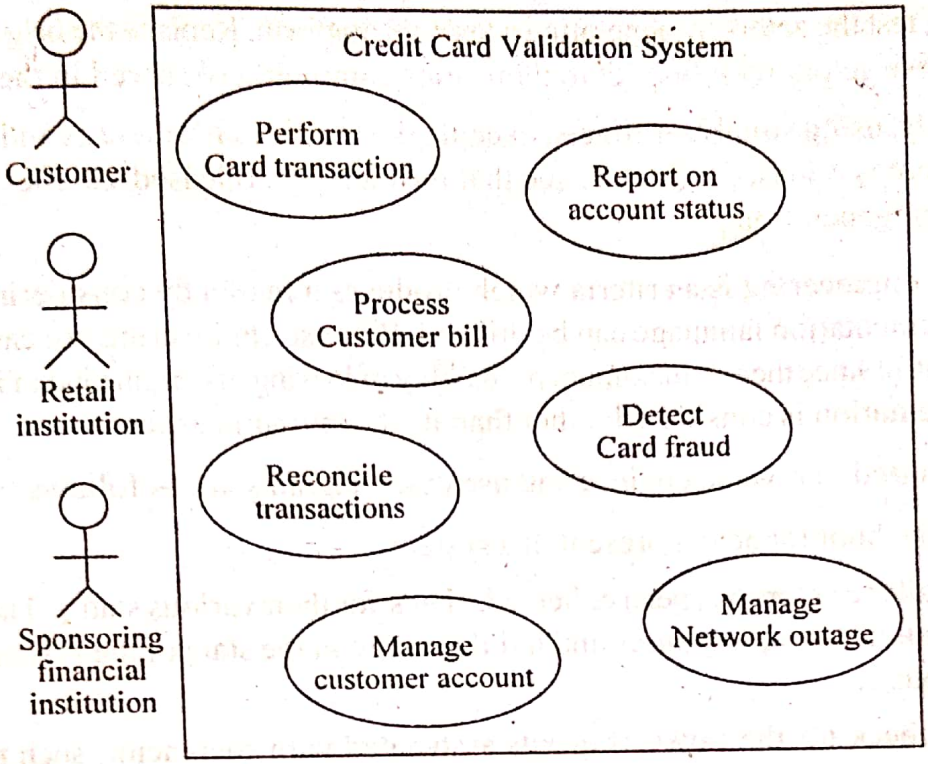


Fig. : Modeling the Requirements of a System

26. What are the steps involved in forward and reverse engineering of use case diagrams ?

Ans :

Forward and Reverse Engineering

Forward engineering is a criteria in which a desired model is mapped into a code. When this process is applied to a use case diagram, the event will not be achieved easily, since use case diagram makes behaviour of an element as it's main focus. Hence, it can be concluded, that, forward engineering use case diagram will reveal a test condition (for every use case present in the diagram) which can be executed as and when a new release of element's behaviour is made. This could assure that the new release works as expected prior to other co-element starts depending on it.

Following are the steps required in forward engineering use case diagrams are as follows :

1. Start the process by initially considering the flow of events and exceptional flow of events associated with every use case in a given diagram.
2. In the next step, generate a test chronicle, for each flow involved in the diagram. Consider the conditions existed prior and the post of the flow. The prior conditions can be taken for testing initial status whereas the post conditions can be taken as a triumph of test chronicle. Here the degree of test chronicle depends on one's desire.

3. Now, test the actors by generating a testable platform. Replace the original identities of these actors with respect to their roles commonly observed in the real world.
4. Finally, using suitable artifacts, execute the testable chronicle as and when a new element is released. Check to see that the elements released satisfies the purpose of their generation.

Reverse engineering is a criteria which produces a model by considering the code. Here an implementation language can be utilized. Reverse engineering use case diagrams will not be fruitful since there is maximum probability of loosing of certain kind of information. Here implementation is considered rather than it's behavioural aspects.

The steps required in reverse engineering use case diagrams are as follows :

1. Initially, hunt for actors present in a system.
2. After all the actors has been collected, check for their various status. This includes their interactions, responses and indulgence with the state of the system in which they exists.
3. Now, check for the flows of events associated with each actor, such that initial preference is given to primary flows followed by the rest.
4. Accumulate the flows which are related to each other into one section. Assign a suitable use case for each section. While doing this, utilize the relationships such as "extend" and "include" respectively. Attach "include" relationship to those positions where there is a modeling of common flows. Similarly, attach "extend" while modeling the variant flows.
5. Finally, consider all the actors and use cases and employ them into proper use case diagram. Use relationships to organize them.

27. What is an activity diagram ? What are its common properties and contents of activity diagram ?

Ans :

Activity Diagram

An activity diagram shows the flow from activity to activity. 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 result in a change in state of the system or the return of a value. Actions encompass calling another operation, sending a signal, creating or destroying an object, or some pure computation, such as evaluating an expression. Graphically, an activity diagram is a collection of vertices and arcs.

We can model these dynamic aspects using activity diagrams, which focus first on the activities that take place among objects, as Figure below shows.

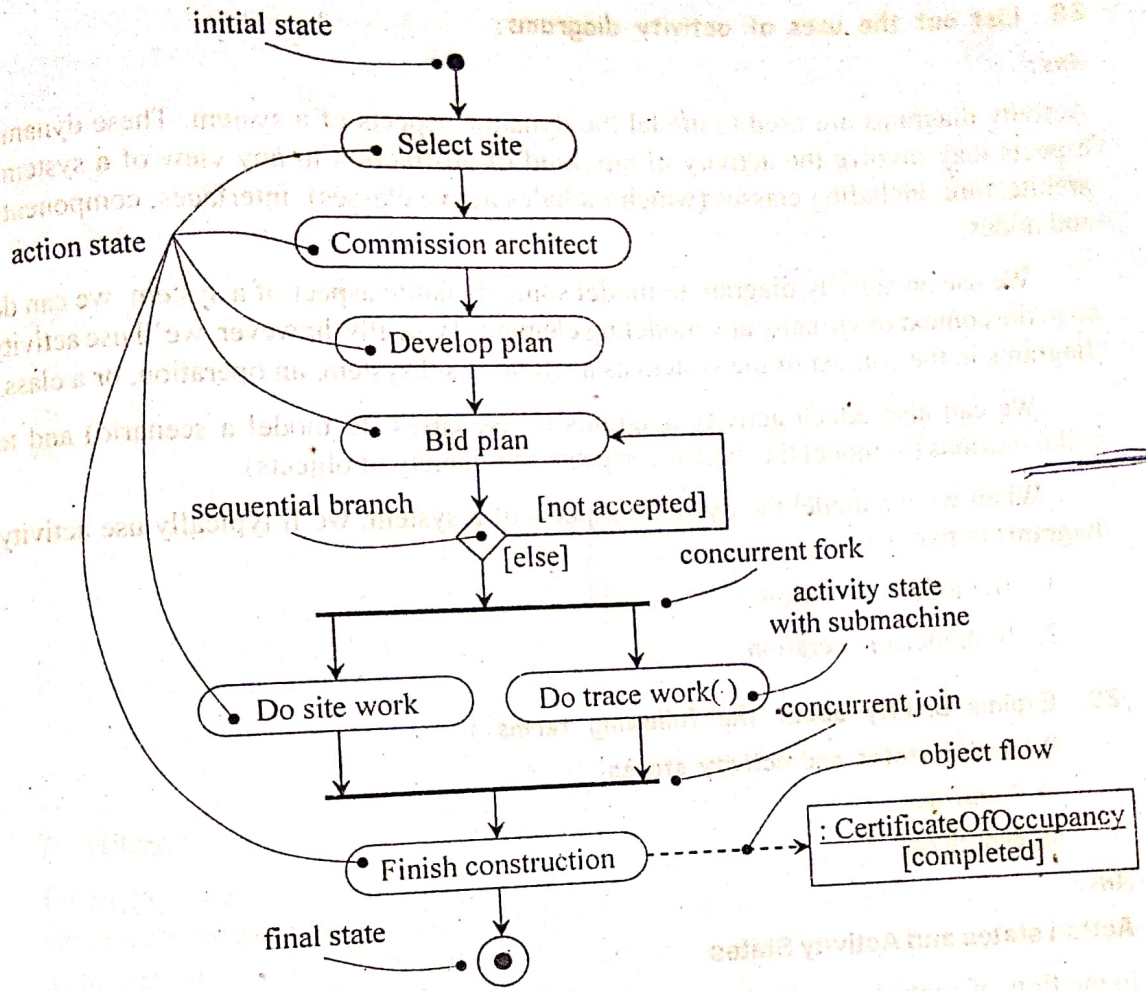


Fig. : Activity Diagrams

An activity diagram is essentially a flowchart that emphasizes the activity that takes place over time. We can think of an activity diagram as an interaction diagram turned inside out. An interaction diagram looks at the objects that pass messages; an activity diagram looks at the operations that are passed among objects. The semantic difference is subtle, but it results in a vary different way of looking at the world.

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.

Activity diagrams commonly contain

- Activity states and action states
- Transitions
- Objects

28. List out the uses of activity diagrams.

Ans :

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

We use an activity diagram to model some dynamic aspect of a system, we can do so in the context of virtually any modeling element. Typically, however, we'll use activity diagrams in the context of the system as a whole, a subsystem, an operation, or a class.

We can also attach activity diagrams to use cases (to model a scenario) and to collaborations (to model the dynamic aspects of a society of objects).

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

1. To model a workflow
2. To model an operation.

29. Explain briefly about the following terms :

i) Action states and activity states

ii) Transitions

iii) Branching

Ans :

Action states and Activity States

In the flow of control modeled by an activity diagram, things happen. We might evaluate some expression that sets the value of an attribute or that returns some value. Alternatively, we 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 below shows, we represent an action state using a lozenge shape (a symbol with horizontal top and bottom and convex sides). Inside that shape, we may write any expression.

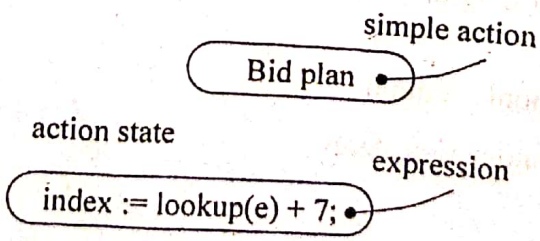


Fig. : Action States

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

Activity states can be further decomposed, their activity being represented by other activity diagrams. Activity states are not atomic, meaning that they may be interrupted and, in general, are considered to take some duration to complete. We can think of an action state as a special case of an activity state. An action state is an activity state that cannot be further decomposed.

As figure below 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.

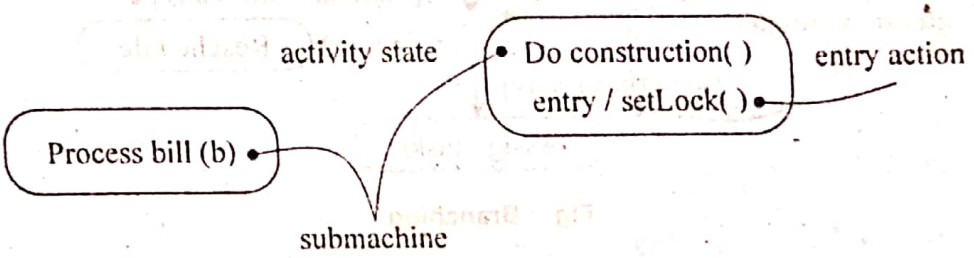


Fig. : Activity States

Transitions

When the action or activity of a state completes, flow of control passes immediately to the next action or activity state. We 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, we represent a transition as a simple directed line, as Figure shows.

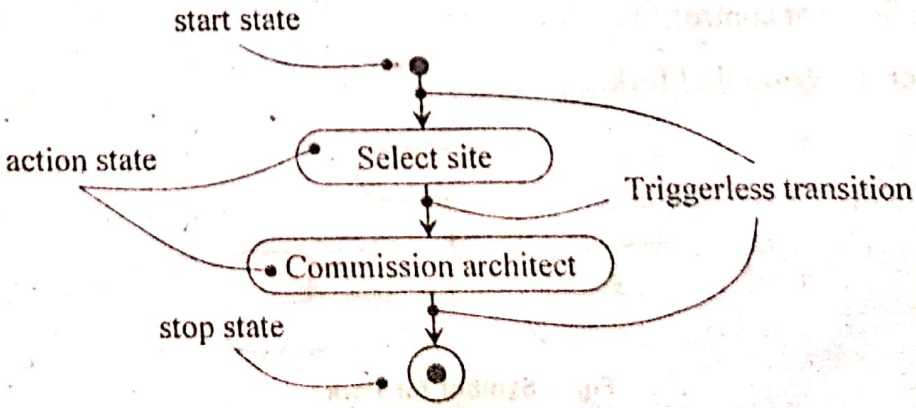


Fig. : Triggerless Transitions

A flow of control has to start and end someplace (unless, of courses, it's an infinite flow, in which case it will have a beginning but no end). Therefore, as the figure shows, you may specify this initial state (a solid ball) and stop state (a solid ball inside a circle).

Branching

As in a flowchart, we can include a branch, which specifies alternate paths taken based on some Boolean expression. As Figure below shows, we represent a branch as a diamond. A branch may have one incoming transition and two or more outgoing one. On each outgoing transition, we place a Boolean expression, which is evaluated only once on entering the branch. Across all these outgoing transitions, guards should not overlap (otherwise, the flow of control would be ambiguous), but they should cover all possibilities (otherwise, the flow of control would freeze).

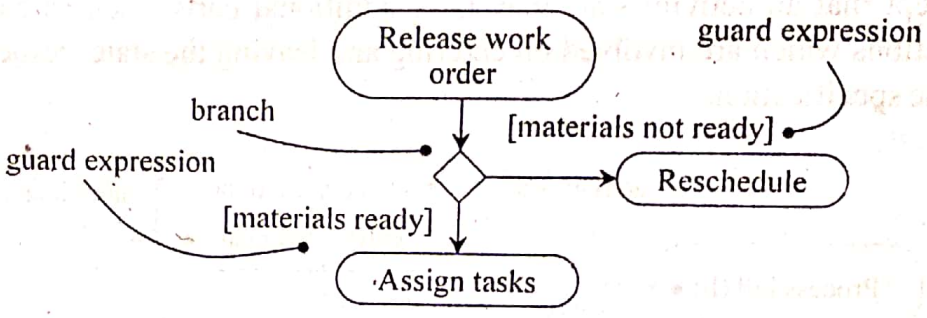


Fig. : Branching

30. Explain how forking and joining is used in activity diagrams.

Ans :

Forking

As the name indicates, forking is a process of splitting a single flow of control into multiple flows of control. Generally a fork has a single incoming flow of control, but multi-outgoing flow of control. The incoming flow of control is separated by outgoing flows of control using a dark horizontal line. The number of outgoing flows of control are termed as independent flows of control.

Consider the symbol of fork.

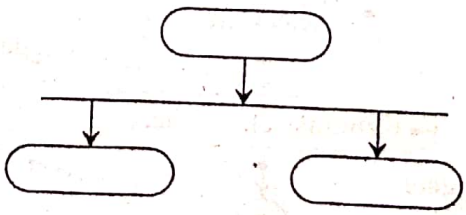


Fig. : Symbol for Fork

Joining

Joining is exactly opposite process of forking i.e., In joining multiple flows of control are clubbed together to give one single independent flow of control. Generally, joining has multiple incoming flows of control, but single outgoing flow of control. Here also, single

dark horizontal line is used to separate the incoming flows of control with o**with an activity** control. The joining is represented by the symbol as shown below.

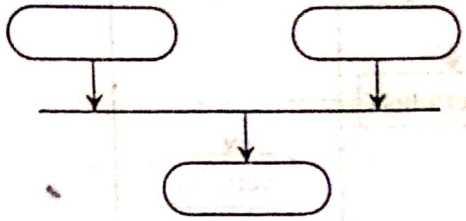
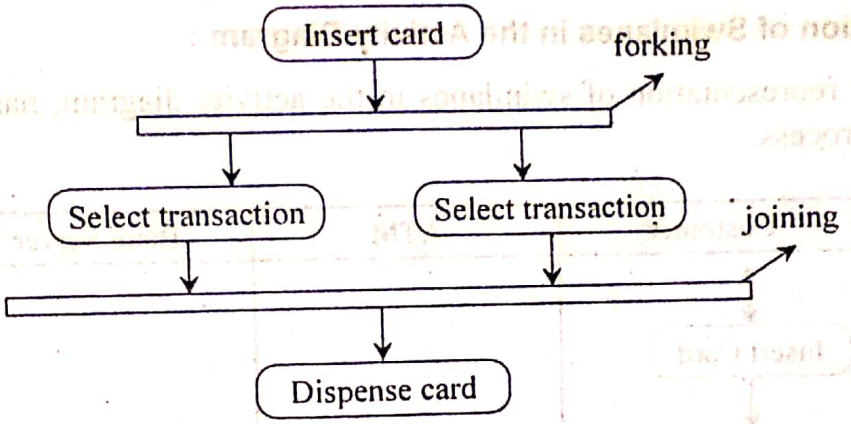


Fig. : Symbol for Join

Example :

Consider the transitions of ATM in brief using forking and joining.



Consider the example given above which represents ATM transaction in brief. Here the ATM user inserts his card (activity) into the ATM machine. This is first transaction, which is forked into two independent activities as "select transaction" and "start transaction". These two activities are joined to form the final activity as the "dispensing the card" (i.e.,) getting the card back from the ATM machine.

31. Write about swimlanes in activity diagrams.

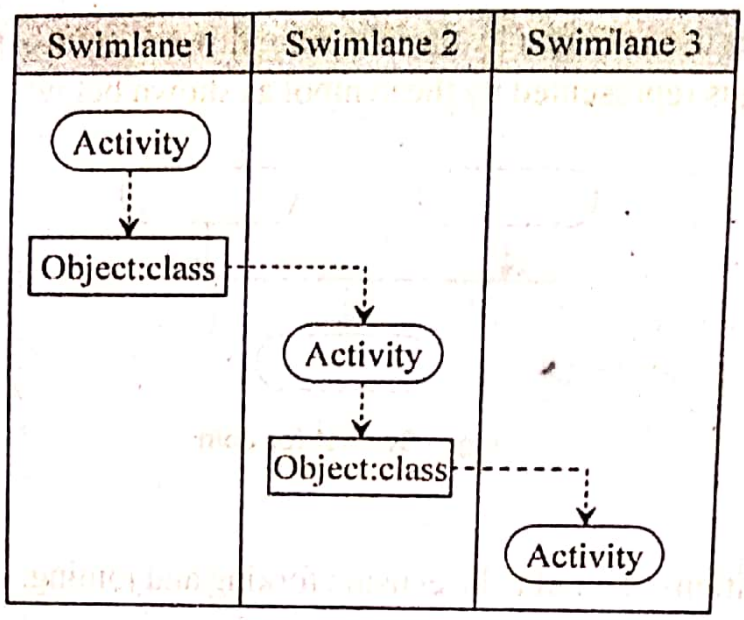
Ans :

Swimlanes

Swimlanes represents the columns in the activity diagrams to group the related activities. These are represented in the form of partitioned regions, where elements (of activity diagrams) participate in various activities and also carries out these activities. Each swimlane has it's name, which is written on the top of the column. The name should be such that, it should relate the activities grouped inside the column and also must be a real world entity.

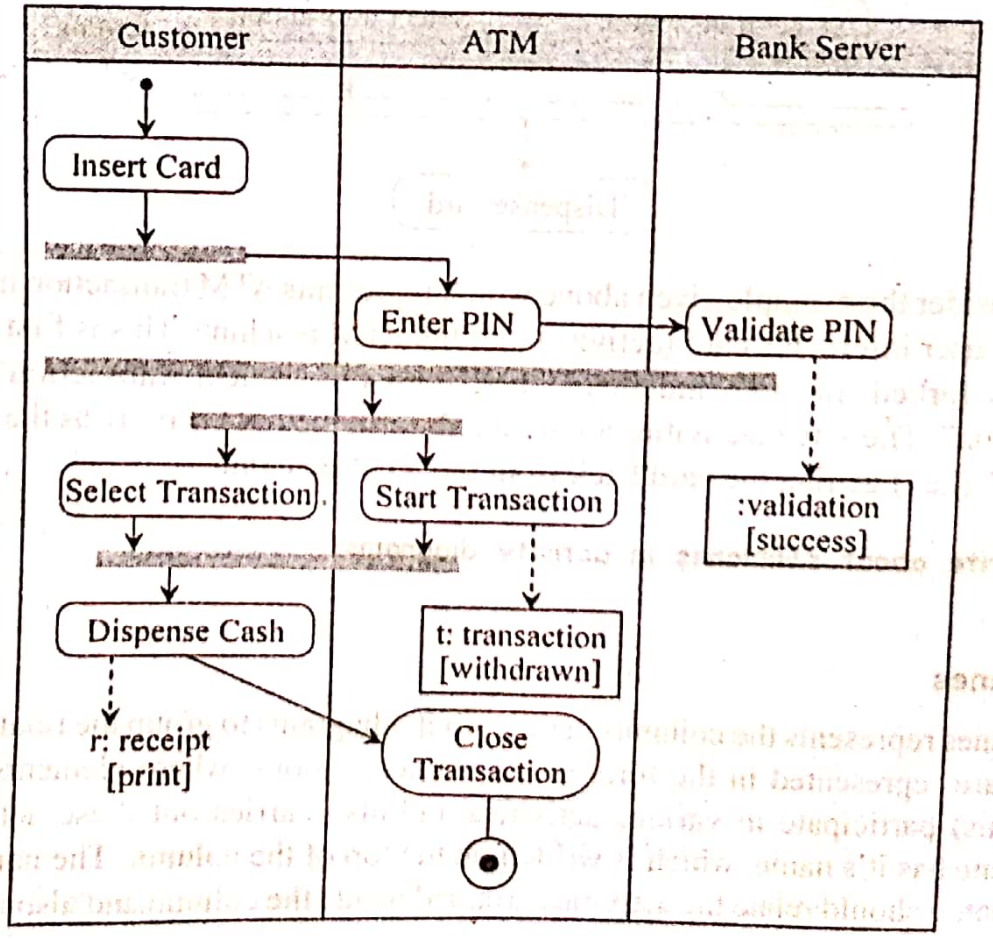
In activity diagrams, the element of each swimlane is restricted to cross the partition, but the transitions (related to these elements) are permitted to flow across the partitions.

Example :



Representation of Swimlanes in the Activity Diagram :

Consider the representation of swimlanes in the activity diagram, narrating the ATM transaction process.



Here the swimlanes are the divisions between the entities customer, ATM and bank server. The various activities performed by these three entities are inscribed within their swimlanes, but at the same time transitions are allowed to cross them.

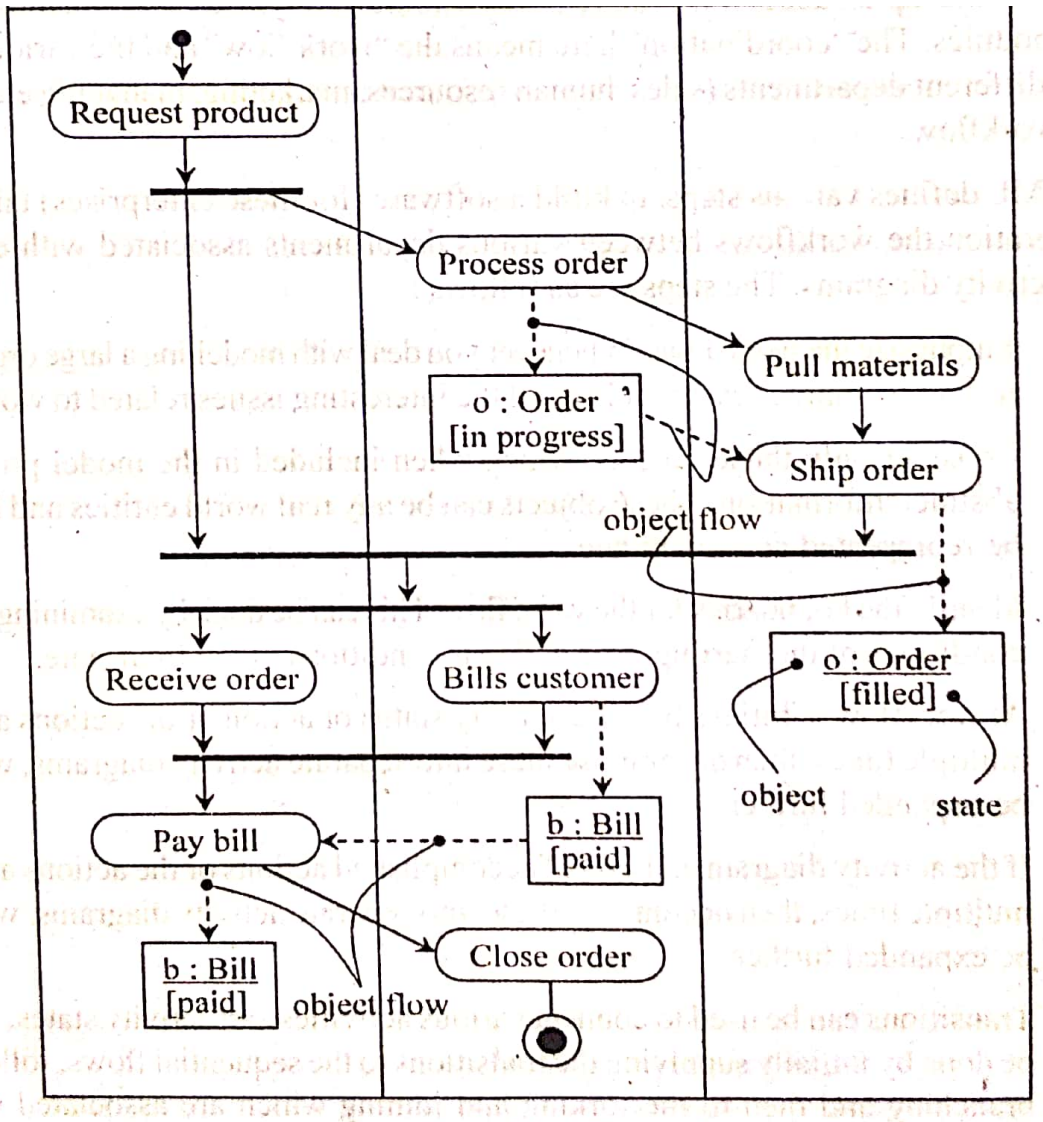


Fig. : Object Flow

As Figure above shows, we can specify the things that are involved in an activity diagram by placing these objects in the diagram, connected using a dependency to the

activity or transition that creates, destroys, or modifies them. This use of dependency relationships and objects is called an object flow because it represents the participation of an object in a flow of control.

As shown in the figure, we represent the state of an object by naming its state in brackets below the object's name. Similarly, we can represent the value of an object's attributes by rendering them in a compartment below the object's name.

33. List out and explain the steps involved in modeling a workflow.

Ans :

Modeling a Workflow

Every large scale enterprises are always divided into different sections or modules. In order to build up a successful product, their should always be "coordination" between these modules. The "coordination" here means the "work flow" and the various modules can be different departments (sales, human resources, marketing, managing etc) involved in the workflow.

UML defines various steps, to build a software (for these enterprises) taking under consideration the workflows between various departments associated with enterprises using activity diagrams. The steps are as follows :

Step 1 : Emphasize the workflows. Whenever you deal with modelling a large organization, it is quite impractical to include all the interesting issues related to workflows.

Step 2 : Consider only those objects, which when included in the model provides the abstract information. These objects can be any real world entities and hence can be represented as a swimlane.

Step 3 : Identify the boundaries for the work flow. This can be done by examining initiating conditions of the starting state and rear conditions of the final state.

Step 4 : At workflow's initial state, use activity status or action or the actions appearing multiple times, then decompose these into separate activity diagrams, which can be expanded further.

Step 5 : If the activity diagram consists of accomplished actions or the actions appearing multiple times, then decompose these into separate activity diagrams, which can be expanded further.

Step 6 : Transitions can be used to connect various activities and activity states. This can be done by initially supplying the transitions to the sequential flows, followed by branching and then to the forking and joining which are associated with the workflows.

Step 7 : In the last step, if there are some important objects which are missed, then include them in the activity diagram by representing its changing values and states.

```
    ... 1.slope) return Point (0, 0) ;
    int x = (1.delta – delta) / (slope – 1.slope) ;
    int y = (slope * x) + delta;
    return Point (x, y);
}
```

Reverse engineering (the creation of a model from code) is also possible for activity diagrams, especially if the 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*.

36. Describe use case "validate user" in modelling an ATM system.

Ans :

Analyzing the Bank ATM System through Use Case Diagrams.

First step is to identify the actors and Use Cases for this ATM System. The actor of the Bank ATM System is the Bank Client. He should be able to deposit an amount, withdraw amount, check amount balance. These activities will be the use cases.

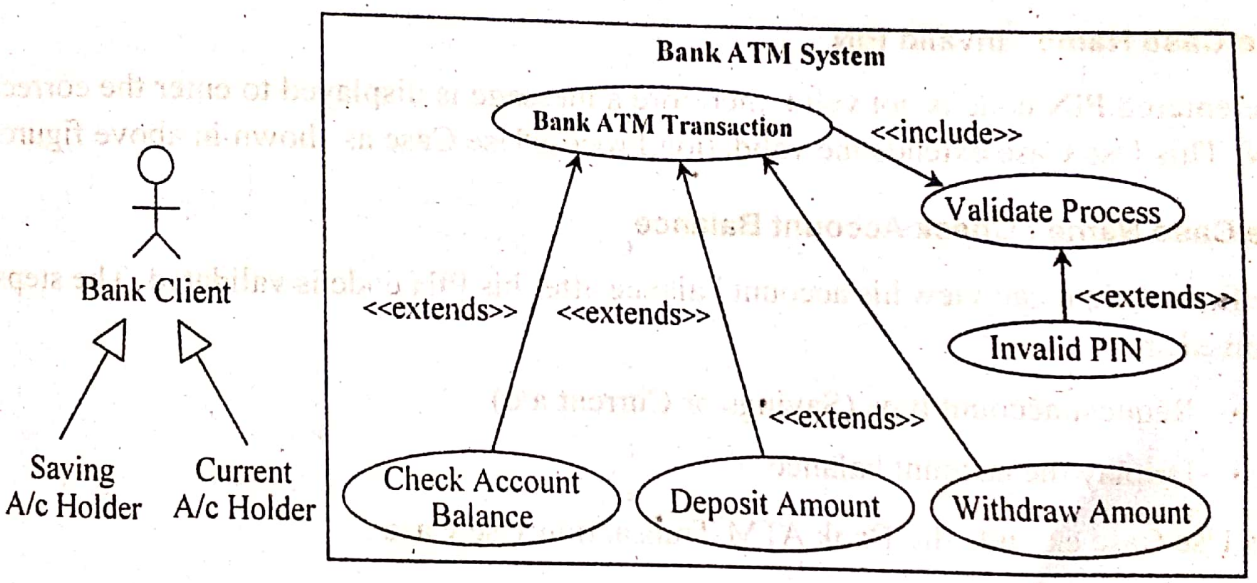


Fig. : Use Case Diagram of a Bank ATM System

Use Case Name : Bank ATM Transaction

The bank client interacts with the bank system by going through the validation process. Only after successful validation the client can perform the transactions. The steps involved in the Bank ATM Transaction Use Case are :

- Insert ATM Card
- Perform Validation
- Request Type of Transaction
- Enter Type of Transaction
- Perform Transaction
- Request Take Card
- Take Card

Use Case Name : Validation Process

A 4 digit PIN Code is entered, if it is valid the account becomes available. The Bank ATM Transaction Use Case includes this (Validation Process) use case and without valid PIN number the client cannot access his/her accounts.

Here are the steps :

- Request PIN code
- Enter PIN code
- Verify PIN code

Use Case Name : Invalid PIN

The entered PIN code is not valid, therefore a message is displayed to enter the correct PIN. This Use Case extends the validation Process Use Case as shown in above figure.

Use Case Name : Check Account Balance

The Bank Client can view his account balance after his PIN code is validated. The steps involved are :

- Request account type (Savings or Current a/c)
- Display the account balance

This Use Case extends the Bank ATM Transaction Use Case

Use Case Name : Deposit Amount

After entering the correct PIN; the Bank Client requests to deposit money to his/her account. The client should select his/her account type and enter the amount in the local currency. This Use Case extends the Bank ATM Transaction Use Case. The steps involved are :

- Request account type
- Request deposit amount
- Enter deposit amount
- Put the cheque or cash in an envelope and insert it into the deposit slit/opening in the ATM.

Use Case Name : Withdraw Amount

When the Bank Client enters the correct PIN, he/she requests to withdraw money from his/her account. After entering the account type, he/she enters the amount to be withdrawn. After verifying from account balance cash is dispensed if funds are sufficient.

- Request account type
- Request withdrawal amount
- Enter withdrawal account
- Verify from account balance for sufficient funds
- Dispense/Eject cash.

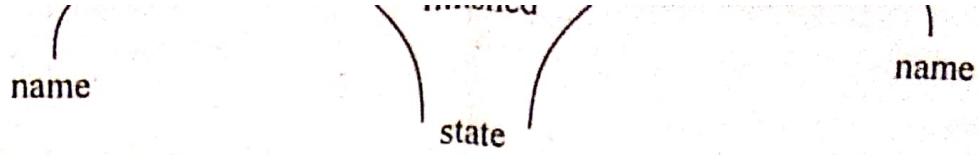


Fig. : States

42. What is a transition ? List out different parts involved in transition.

Ans :

Transitions

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. For example, a *Heater* might transition from the *Idle* to the *Activating* state when an event such as *tooCold* (with the parameter *desiredTemp*) occurs.

A transition has five parts :

1. ***Source state :*** The state affected by the transition, if an object is in the source state, an outgoing transition may fire when the object receives the trigger event of the transition and if the guard condition, if any, is satisfied.
2. ***Event trigger :*** The event whose reception by the object in the source state makes the transition eligible to fire, providing its guard condition is satisfied.
3. ***Guard condition :*** A Boolean expression that is evaluated when the transition is triggered by the reception of the event trigger, if the expression evaluates True, the transition is eligible to fire; if the expression evaluates False, the transition does not fire and if there is no other transition that could be triggered by that same event, the event is lost.
4. ***Action :*** An executable atomic computation that may directly act on the object that owns the state machine, and indirectly on other objects that are visible to the object.
5. ***Target state :*** The state that is active after the completion of the transition.

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

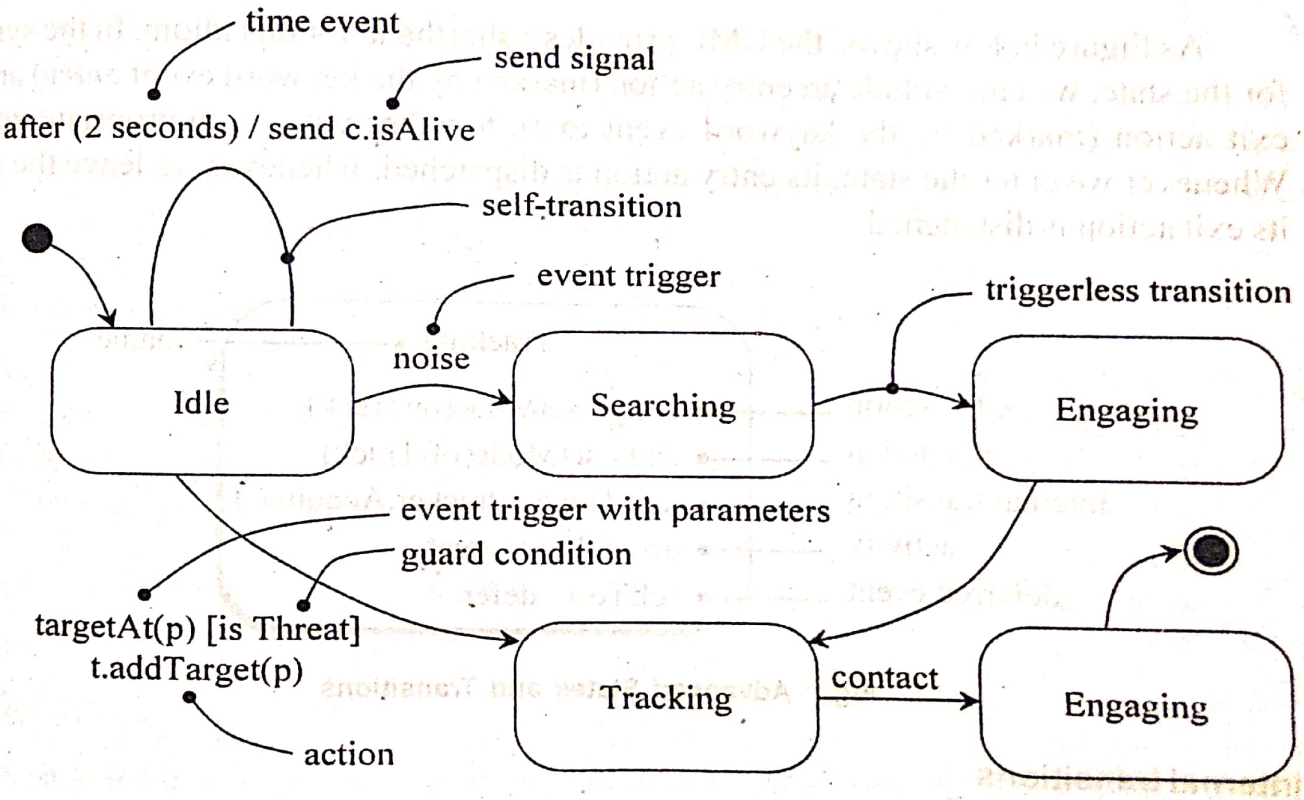


Fig. : Transitions

46. Write about active classes.

Ans :

Active Classes

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.

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.

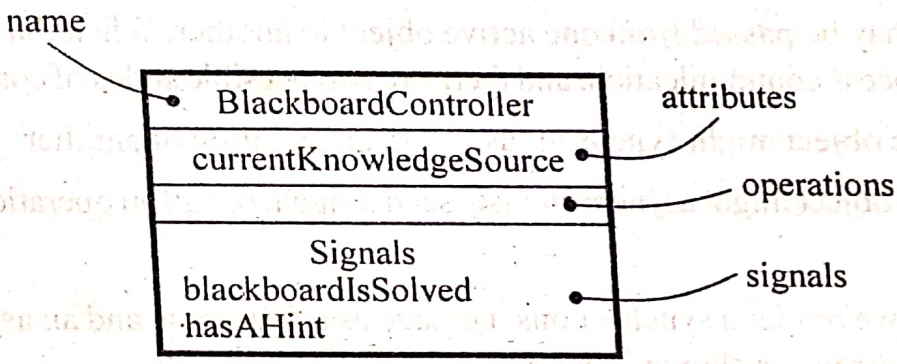


Fig. : Active Class

47. What are the two standard stereotypes that apply to active classes ?

Ans :

The UML defines two standard stereotypes that apply to active classes.

1. **Process :** Specifies a heavyweight flow that can execute concurrently with other processes.
2. **Thread :** Specifies a lightweight flow that can execute concurrently with other threads within the same process.

Process

A process is heavyweight, which means that it is a thing known to the operating system

UNIT - 3

OBJECT ORIENTED SYSTEM DEVELOPMENT

Architectural Modeling : Artifacts, Deployment, Collaborations, Patterns and Frameworks, Deployment diagrams, Component Diagrams, Systems and Models.

1. What is a component ? Give the graphical representation of a component. What are the properties of components ?

Ans :

Components

A component is a physical and replaceable part of a system that conforms to and provides the realization of a set of interfaces. Graphically, a component is rendered as a rectangle with tabs.

The UML provides a graphical representation of a component, as Figure below . This canonical notation permits us to visualize a component apart from any operating m or programming language.

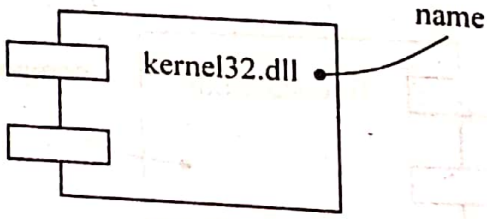


Fig. Component

Every component must have a name that distinguishes it from other components. A *name* is a textual string. That name alone is known as a *simple name*; a *path name* is the component name prefixed by the name of the package in which that component lives.

Properties of Components

Components Possess four important properties, they are :

1. Physical in nature i.e., components should be referred to documents consisting of data in the form of text, not to any conceptual facts.
2. *Replaceable* It means that components can be replaced by any other component, provided it should satisfy the existing situation. This causes no loss of information rather sometimes enhances the capability of the system.
3. Internal entity of the system. This means that components always collaborate to other components (existing in the system) to provide a well defined structure to the system. It is basic entity of a system rather than a system on a whole.
4. Finally components can be associated to a set of interfaces.

1. What is a component ? Give the graphical representation of a component. What are the properties of components ?

Ans :

Components

A component is a physical and replaceable part of a system that conforms to and provides the realization of a set of interfaces. Graphically, a component is rendered as a rectangle with tabs.

The UML provides a graphical representation of a component, as Figure below shows. This canonical notation permits us to visualize a component apart from any operating system or programming language.

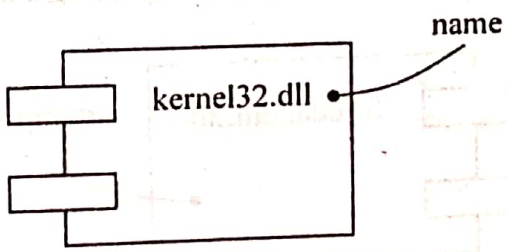


Fig. Component

Every component must have a name that distinguishes it from other components. A *name* is a textual string. That name alone is known as a *simple name*; a *path name* is the component name prefixed by the name of the package in which that component lives.

Properties of Components

Components Possess four important properties, they are :

1. Physical in nature i.e., components should be referred to documents consisting of data in the form of text, not to any conceptual facts.
2. *Replaceable* It means that components can be replaced by any other component, provided it should satisfy the existing situation. This causes no loss of information rather sometimes enhances the capability of the system.
3. Internal entity of the system. This means that components always collaborate to other components (existing in the system) to provide a well defined structure to the system. It is basic entity of a system rather than a system on a whole.
4. Finally components can be associated to a set of interfaces.

2. What are the significant differences between components and classes ?

Ans :

Components and Classes

Components are like classes : Both have names; both may realize a set of interfaces; both may participate in dependency, generalization, and association relationships; both may be nested; both may have instances; both may be participants in interactions. There are some significant differences between components and classes.

1. Classes represent logical abstractions; components represent physical things that live in the world of bits.
2. Components represent the physical packaging of otherwise logical components and are at a different level of abstraction.
3. Classes may have attributes and operations directly. Components only have operations that are reachable only through their interfaces.

A component is the physical implementation of a set of other logical element, such as classes and collaborations. As Figure below shows, the relationship between a component and the classes it implements can be shown explicitly by using a dependency relationship.

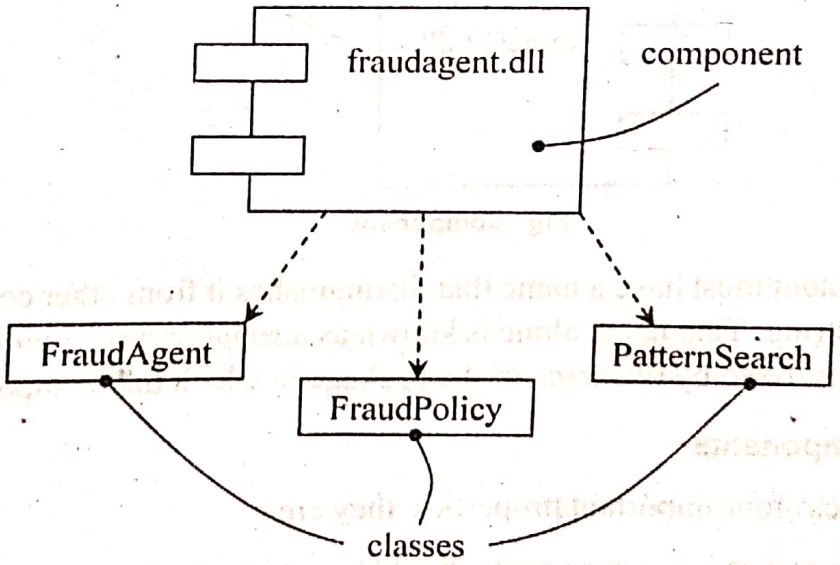


Fig. : Components and Classes

3. Explain about components and interfaces.

Ans :

Components and Interfaces

An interface is a collection of operations that are used to specify a service of a class or a component. The relationship between component and interface is important.

In figure below, we can see the relationship between a component and its interfaces in one of two ways.

1) renders the interface in its elided, iconic form.

The component that realizes the interface is connected to the interface using an elided realization relationship.

2) renders the interface in its expanded form, perhaps revealing its operations.

The component that realizes the interface is connected to the interface using a full relaxation relationship. In both cases, the component that accesses the services of the other component through the interface is connected to the interface using a dependency relationship.

An interface that a component realizes is called an export interface, meaning an interface that the component provides as a service to other components. A component may provide many export interfaces.

The interface that a component uses is called an import interface, meaning an interface that the component conforms to and so builds on. A component may conform to many import interfaces. A component may both import and export interfaces.

A component that uses a given interface will function properly no matter what component realizes that interface. A component can be used in a context if and only if all its import interfaces are provided by the export interfaces of other components.

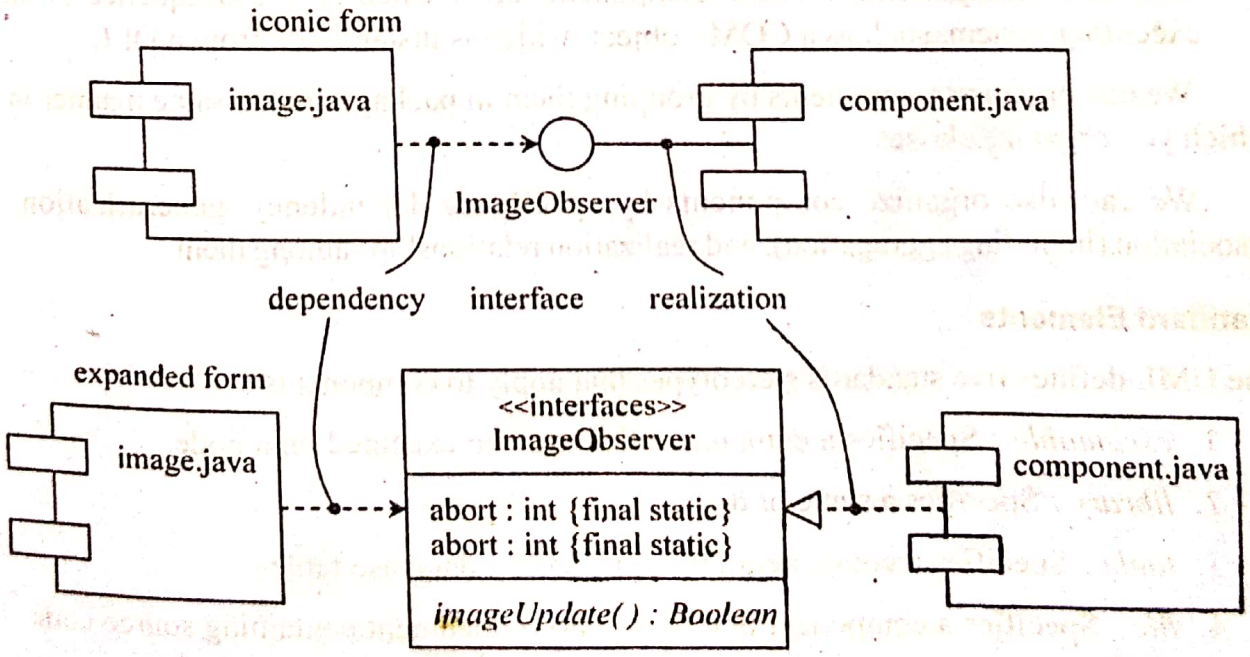


Fig. : Components and Interfaces

4. What are the different kinds of components ? What are the standards stereotypes that apply to components ?

Ans :

Kinds of Components

The three kinds of components are :

1. Deployment components
2. Work product components
3. Execution components

1. ***Deployment components :*** These are the components necessary and sufficient to form an executable system, such as dynamic libraries (DLLs) and executables (EXEs). The UML's definition of component is broad enough to address classic object models, such as COM+, CORBA, and Enterprise Java Beans, as well as alternative object models, perhaps involving dynamic Web pages, database tables, and executables using proprietary communication mechanisms.
2. ***Work product components :*** These components are essentially the residue of the development process, consisting of things such as source code files and data files from which deployment components are created. These components do not directly participate in an executable system but are the work products of development that are used to create the executable system.
3. ***Execution components :*** These components are created as a consequence of an executing system, such as a COM+ object, which is instantiated from a DLL.

We can organize components by grouping them in packages in the same manner in which you organize classes.

We can also organize components by specifying dependency, generalization, association (including aggregation), and realization relationships among them.

Standard Elements

The UML defines five standards stereotypes that apply to components :

1. *executable :* Specifies a component that may be executed on a node.
2. *library :* Specifies a static or dynamic object library.
3. *table :* Specifies a component that represents a database table
4. *file :* Specifies a component that represents a document containing source code or data.
5. *document :* Specifies a component that represents a document.

5. Write the steps to model executables and libraries.

Ans :

The following are the steps with are essential while modeling executables and libraries.

1. Initially consider the aspects in favour of splitting a given system. While doing this, the issues such as reuse, technical and configuration management should be taken into consideration.
2. Now, consider the artifacts (i.e., executables and libraries) and adhere them with components. While doing so, there is a possibility that several new components can come into being. Manage these components by adorning them with suitable extensible mechanism like "stereotypes".
3. If certain issues like "managing seams" becomes necessary, then, it is suggested to model the interfaces that might be applicable with a given component and can be further realized by other co-entities.
4. Finally, use dependency relationships among interfaces libraries and executable before ending your task.

Example : Figure below shows a set of components drawn from a personal productivity tool that runs on a single personal computer. This figure includes one executable (*animator.exe*, with a tagged value noting its version number) and four libraries (*dlog.dll*, *wrfrme.dll*, *render.dll*, and *raytrce.dll*), all of which use the UML's standard elements for executables and libraries, respectively. This diagram also presents the dependencies among these components.

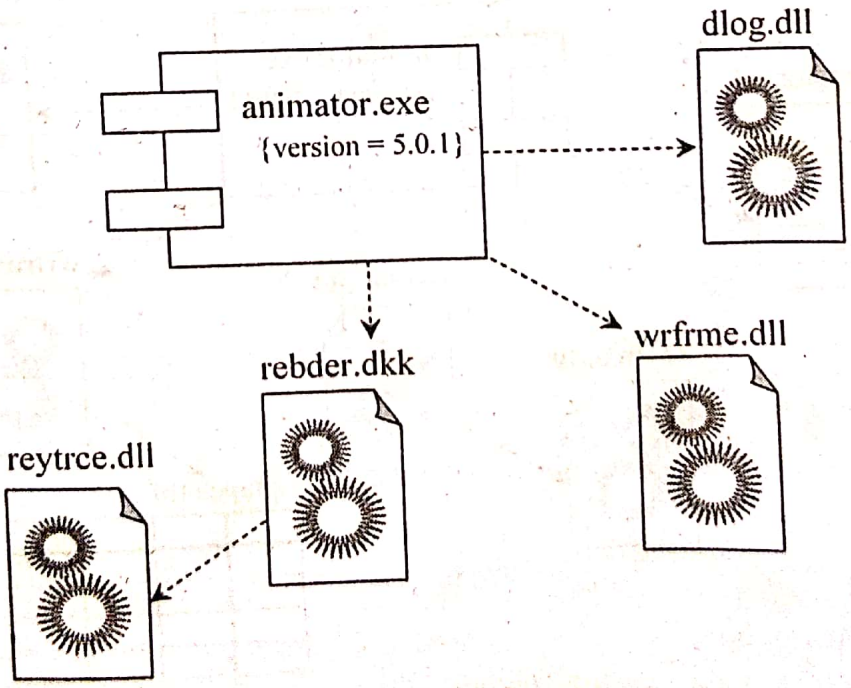


Fig. : Modeling Executables and Libraries

6. Enumerate the steps to model tables, files and documents.

Ans :

Following are the steps to model tables, files, and documents :

1. Identify the ancillary components that are part of the physical implementation of the system.
2. Model these things as components. If the implementation introduces new kinds of artifacts, introduce a new appropriate stereotype.
3. An necessary to communicate the intent, model the relationships among these ancillary components and the other executables, libraries, and interfaces in the system. Most often, we'll want to model the dependencies among these parts in order to visualize the impact of change.

Example : Figure below builds on the previous figure and shows the tables, files, and documents that are part of the deployed system surrounding the executable *animator.exe*. This figure includes one document (*animator.hlp*), one simple file (*animator.ini*), and one database table (*shapes.tbl*), all of which use the UML's standard elements for documents, files, and tables, respectively.

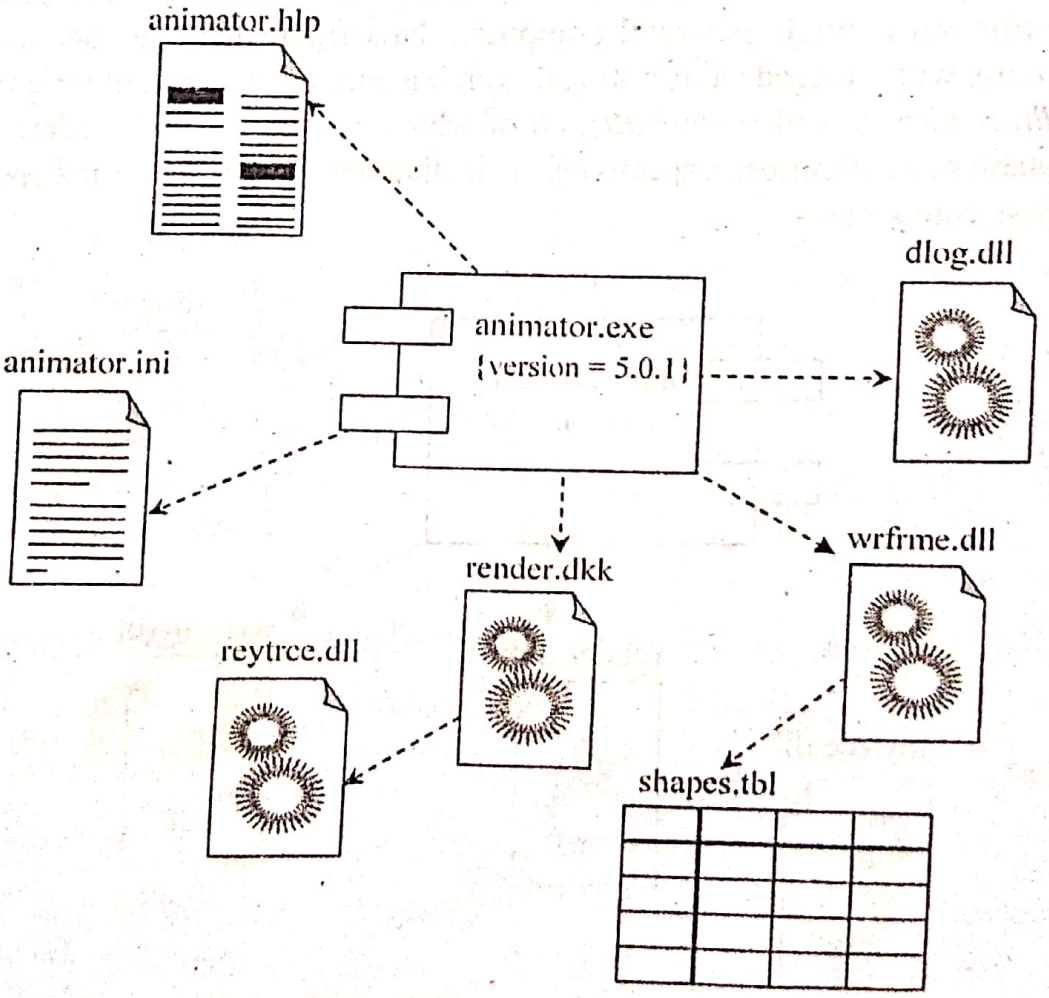


Fig. : Modeling Tables, Files, and Documents

7. List out steps involved in modeling an API.

Ans :

Following are the steps involved in modelling APIs are as follows :

1. Begin the process by initially considering programmatic seams in a given system to be modelled. Adhere these seams by suitable interfaces.
2. It is optional whether to expose/hide properties of interface. If exposure is to be made, then reflect only the properties which can increase the visuality of a given context and if these properties are to be hidden them document them so that it can be referred later.
3. Finally, end up the session by considering the realization relationship to each API.

The four interfaces that form the API of the executable are : *IApplication*, *IModel*, *IRendering*, and *IScripts*.

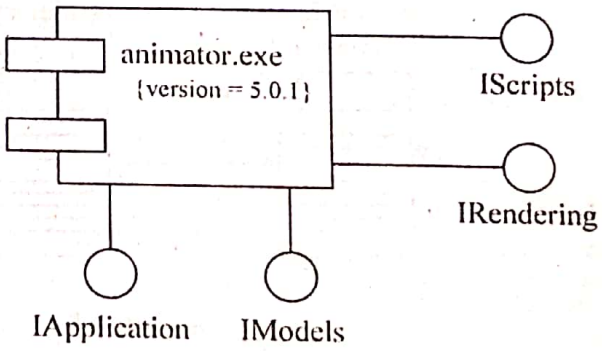


Fig. : Modeling an API

8. Enumerate the steps to model source code.

Ans :

Modeling Source Code

Modeling source code graphically is particularly useful for visualizing the compilation dependencies among the source code files and for managing the splitting and merging of groups of these files when we form and join development paths.

Source code files are drawn from the decisions we make about how to segment the files our development environment needs. These files are used to store the details of our classes, interfaces, collaborations, and other logical elements as an intermediate step to creating the physical, binary components that are derived from these elements by your tools.

To model source code, following steps are be required :

1. Depending on the constraints imposed by your development tools, model the files used to store the details of all your logical elements, along with their compilation dependencies.

2. If it's important for you to bolt these models to your configuration management and version control tools, you'll want to include tagged values, such as version, author, and check in/check out information, for each file that's under configuration management.
3. As far as possible, let your development tools manage the relationships among these files, and use the UML only to visualize and document these relationships.

Example :

Figure below shows some source code files that are used to build the library *render.dll* from the previous examples. This figure includes four header files (*render.h*, *rengine.h*, *poly.h*, and *colortab.h*) that represent the source code for the specification of certain classes. There is also one implementation file (*render.cpp*) that represents the implementation of one of these header.

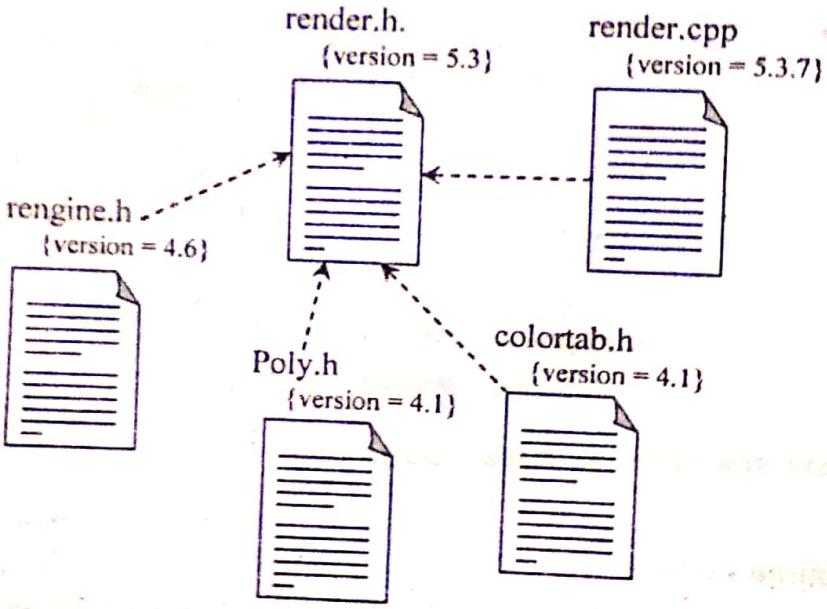


Fig. : Modeling Source Code

9. **What are the common properties, contents and uses of component diagrams ?**

Ans :

Component Diagrams

Component diagrams are basically used to model static view of a system. This can be achieved by modelling various physical components like libraries, tables, files etc. which are residing internal to given node (where node can be any physical identity which can promote large number of executions parallely *eg.* : hub, server etc.).

Component diagrams are very essential for constructing executable systems. This can be done using concepts of forward and reverse engineering.

Contents of Component Diagram

Component diagrams commonly contain :

1. Components
2. Interfaces
3. Relationships

1. ***Components :*** Components refer to the part or element of the system to be modelled. This can be executables, libraries, tables, files etc. Components also binds several classes, interfaces and collaborations by physical representation.
2. ***Interfaces :*** Interfaces consists of operations that belongs to classes or components residing in the models. It gives the external behaviour of a component to which it is attached. The behaviour can be complete or partial. Therefore interface specifies set of activities, but it (interface) does not account for implementation of these activities.
3. ***Relationships :*** Components diagrams supports four types of relationships.
   i) *Dependency :* It is the relationship between the two components in which change made to one component may exert its impact on the other component.
   ii) *Association :* It is the relationship between the two components which binds them (components) to one particular location.
   iii) *Generalization :* It is the relationships in which objects of one component can be replaced with the objects of another component.
   iv) *Realization :* It is the relationship in which a component specifies the services which are carried out by another component. Component diagrams may also include packages, according to the requirement of the systems. These packages are essential to club all the components to form one large component.

Common Properties of Component Diagrams

The contents component diagrams reflects its properties, since using these artifacts we can easily distinguish component diagrams with various other UML supportive diagrams.

Uses of Component Diagrams

Whenever there is a requirement of modelling the physical aspects of object-oriented systems, then component diagrams are termed to be the best option.

The uses of component diagrams are as follows :

1. They are extensively used in modelling the adaptable systems i.e., the system which relocates itself during failures. These systems also consists of some mobile agents which shifts its location depending on the situations i.e., shifting occurs whenever there is a need to balance the load or during crash recovery.

2. Component diagrams also serves best when used in modelling executable releases.
3. Modern way of storing the information on the systems is by using physical databases. These databases stores information using tables or pages. Thus, component diagrams provides good results even in modelling the physical databases.
4. Whenever a program is written using any programming language, the source code of the program is stored in the form of files. Component diagrams models can be used to model this source code more effectively.

10. Explain how to model an

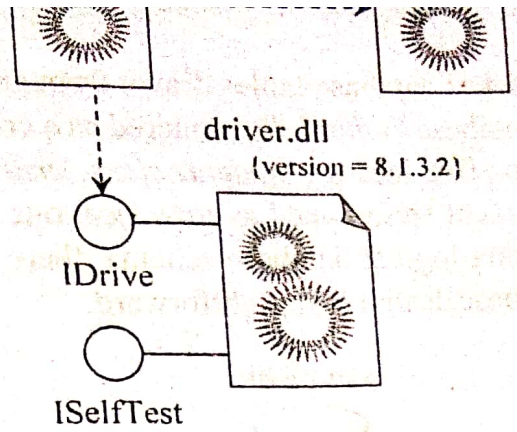


Fig. : Modeling an Executable Release

11. Enumerate the steps to model physical database.

Ans :

UML provides many mechanisms to model the physical database schema. The following are the steps essential in this purpose.

1. While modelling physical database we come across many classes. Define tables for each of these classes for simplicity.
2. Break the inheritance lattices, to align the instances defined by a class on the same state.
3. Make necessary arrangement to differentiate the parent and child states. Insert them into separated tables.

Modeling physical database schema requires mapping of elements which are included in the logical database schema. Here are some of the commitments which are essential while implementing them. They are :

1. Use SQL for implementing the operations like delete, read, update, create etc.
2. While dealing with complex behaviours, apply stored procedures to attain best results in mapping.

Basing on the above statements, here are some of the conclusions which are also essential while modeling physical database.

1. Select the classes forming the constituents of logical database schema.
2. By applying different mechanisms, map the classes to their respective tables.
3. Use stereotypes in the form of tables and apply them in the evolved component diagram which is essential to visualize, construct, specify and document the mapping.

4. Thus translate the logical design into a physical design using appropriate tools.

Hence these were the essentials which are required while modelling a physical database schema.

Figure below shows a set of database tables drawn from an information system for a school. We will find one database (*school.db*, rendered as a component stereotyped as *database*) that's composed of five tables : *student*, *class*, *instructor*, *department*, and *course* (rendered as a component stereotyped as *interface*, one of the UML's standard elements). In the corresponding logical database schema, there was no inheritance, so mapping to this physical database design is straightforward.

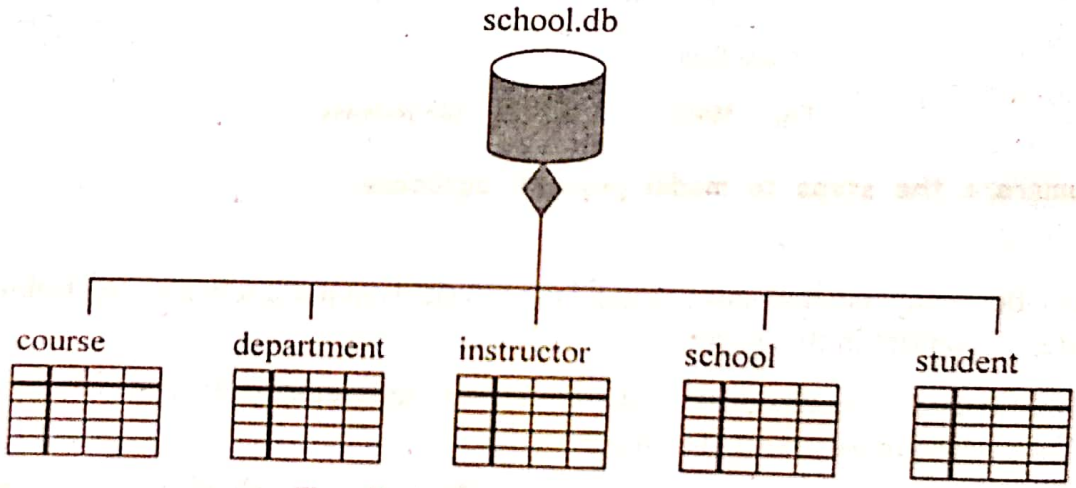


Fig. : Modeling a Physical Database

12. List out and explain the steps involved in modeling adaptable systems.

Ans :

Component diagrams are generally used to model the physical aspects of any object oriented systems. Here, the physical aspects refers to the servers or any storage devices or any databases, etc. Modelling these elements refers to the static view of the system. But the main quality of component diagrams are judged only when they are used to model the complex "Adaptable system", i.e., the systems which dynamically relocates itself to another systems, during the failure of the existing systems. The various steps provided by the UML to model adaptable systems are as follows :

Step 1 : Initially, identify those components which relocates itself between different nodes depending on the situation (modelling events).

Step 2 : Specify location of these nodes using tagged values and implement them in the component diagram.

Step 3 : To make model more lively use interaction diagrams to model the causes which made the action of migration of nodes possible.

17. What are the common properties, contents and uses of deployment diagrams ?

Ans :

Deployment Diagrams

A deployment diagram is a diagram that shows the configuration of run time processing nodes and the components that live on them. Graphically, a deployment diagram is a collection of vertices and arcs.

Common Properties

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

Contents

Deployment diagrams commonly contain

1. Nodes
2. Dependency and association relationships

Deployment diagrams may contain notes and constraints. Deployment diagrams may also contain components, each of which must live on some node. Deployment diagrams may also contain packages or subsystems, both of which are used to group elements of your model into larger chunks.

We use deployment diagram to model the static deployment view of a system. This view primarily addresses are distribution, delivery, and installation of the parts that make up the physical system.

There are some kinds of systems for which deployment diagrams are unnecessary. If we are developing a piece of software that lives on one machine and interfaces only with standard devices on that machine that are already managed by the host operating system (for example, a personal computer's keyboard, display, and modem), we can ignore deployment diagrams.

When you model the static deployment view of a system, we typically use deployment diagrams in one of three ways :

1. To model embedded systems
2. To model client/server systems
3. To model fully distributed systems

We can use deployment diagrams to visualize the system's current topology and distribution of components to reason about the impact of changes on that topology.

18. Enumerate the steps to model embedded system.

Ans :

Modeling Embedded System

Deployment diagrams are useful in facilitating the communication between the project's hardware engineers and software developers. Deployment diagrams are also helpful in reasoning about hardware/software trade-offs. We'll use deployment diagrams to visualize, specify, construct, and document your system engineering decisions.

Steps to model an embedded system :

1. Identify the devices and nodes that are unique to the system.
2. Provide visual cues, especially for unusual devices, by using the UML's extensibility mechanisms to define system - specific stereotypes with appropriate icons.
3. Model the relationships among these processors and deviced in a deployment diagram. Similarly, specify the relationship between the components in the system's implementation vies and the nodes in the system's deployment view.
4. As necessary, expand on any intelligent devices by modeling their structure with a more detailed deployment diagram.

19. Write the steps involved in modeling a client/server system.

Ans :

Modeling a Client/Server System

We can use the UML's deployment diagrams to visualize, specify, and document our decisions about the topology of our client/server system and how its software components are distributed across the client and server.

Following are the steps to model a client/server system :

1. Identify the nodes that represent the system's client and server processors.
2. Highlight those devices that are germane to the behaviour of your system. For example, if we want to model special devices, such as credit card readers, budget readers, and display devices other than monitors, because their placement in the system's hardware topology are likely to be architecturally significant.
3. Provide visual causes for these processors and devices via stereotyping.
4. Model the topology of these nodes in a deployment diagram. Similarly, specify the relationship between the components in the system's implementation view and the nodes in the system's deployment view.

Example :

Figure below, shows the topology of a human resources system, which follows a classical client/server architecture. Figure illustrates the client/server split explicitly by using the

packages named *client* and *server*. The client package contains two nodes (*co kiosk*), both of which are stereotyped and are visually distinguishable. The serve contains two kinds of nodes (*caching server* and *server*), and both of these h adorned with some of the components that reside on each. *Caching server* an are marked with explicit multiplicates, specifying how many instances of each are expected in a particular deployed configuration. For example, this diagram indicates that there may be two or more *caching servers* in any deployed instance of the system.

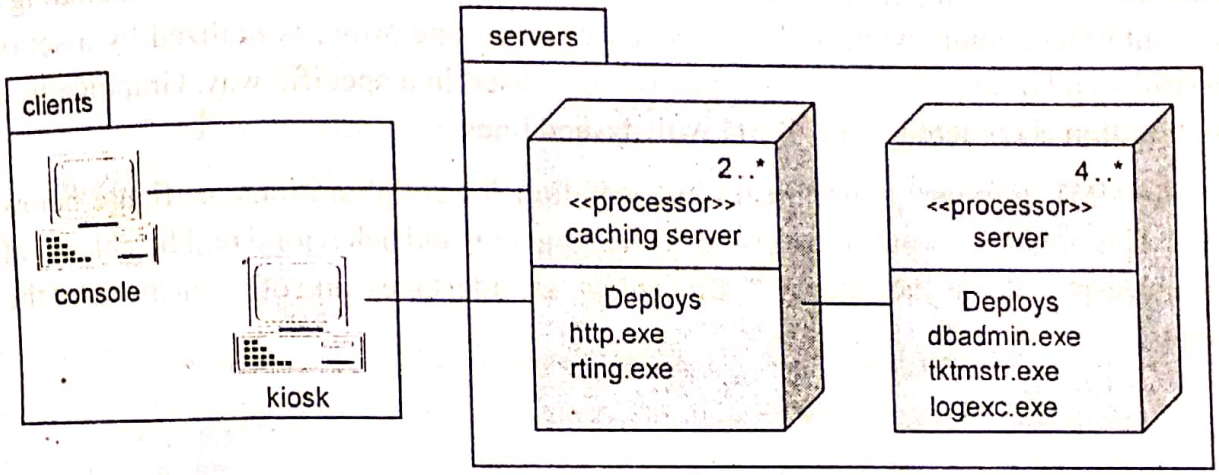


Fig. : Modeling a Client/Server System

20. Explain about the reverse engineering a deployment diagram.

Ans :

Reverse Engineering

Whenever deployment diagrams are used in modelling object-oriented systems, they provide well structured features to represent their physical aspects. These diagrams possess some of the features of class diagrams and are effective especially in modelling different system's nodes.

Here are the following steps, which are essential in reverse engineering deployment diagram.

1) The first step is selection of the node.
2) According to the consistency of the approach, reverse engineer either the system's processor or networking peripherals.
3) After selecting nodes, using desires tools, recover different elements which work parallely with the nodes. Introduce these elements in the deployment diagram.
4) Lastly, by examining the existing model, evaluate the deployment diagrams. If any component requires certain details of it to be made available, then expose (or) If any component possess details which are non necessary, drop it.

21. Define a Collaboration. Explain the importance of collaboration in architectural modeling.

Ans :

Collaborations

A *collaboration* is a society of classes, interfaces, and other elements that work together to provide some cooperative behavior that's bigger than the sum of all its parts. A collaboration is also the specification of how an element, such as a classifier (including a class, interface, component, node, or use case) or an operation, is realized by a set of classifiers and associations playing specific roles used in a specific way. Graphically, a collaboration is rendered as an ellipse with dashed lines.

The UML provides a graphical representation for collaborations, as figure below shows. This notation permits us to visualize the structural and behavioral building blocks of a system, especially as they may overlap the classes, interfaces, and other elements of the system.

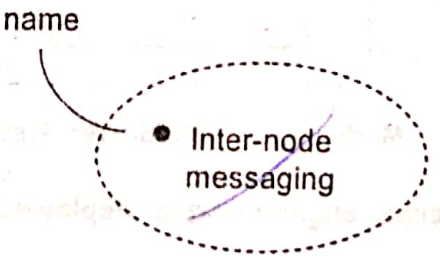


Fig.: Collaborations

A collaboration gives a name to the conceptual building blocks of the system, encompassing both structural and behavioral elements. For example, we might have a distributed management information system whose database are spread across several nodes. For the user's perspective, updating information looks atomic; from the inside perspective, it's not so simple, because such an action has to touch multiple machines.

To give the illusion of simplicity, we'd want to devise a transaction mechanism with which a client could name what looks like a single, atomic transaction, even across various databases. Such a mechanism would span multiple classes working together to carry out a transaction. Many of these classes would be involved in other mechanisms as well, such as mechanisms for making information persistent. This collection of classes (the structural part), together with their interactions (the behavioral part), forms a mechanism, which, in the UML, we can represent as a collaboration.

Collaborations not only name a system's mechanisms, they also serve as the realization of use cases and operations.

22. What are the rules to be followed to give a name to collaboration ?

Ans :

Names

1) Every collaboration must have a name that distinguishes it from other collaborations.
2) A *name* is a textual string. That name alone is known as a *simple name*;
3) A *path name* is the collaboration name prefixed by the name of the package in which that collaboration lives.
4) Typically, a collaboration is drawn showing only its name, as in the previous figure.

A collaboration name may be text consisting of any number of letter, numbers and certain punctuation marks and may continue over several lines.

23. Explain two aspects of the Collaboration.

or

Explain the structural and Behavioral aspects of the Collaboration.

Ans :

Collaborations have two aspects :

1) ***Structural part :*** It specifies the classes, interfaces, and other elements that work together to carry out the named collaboration, and
2) ***Behavioral part :*** It specifies the dynamics of how those elements interact.

Structural Aspect

The structural part of a collaboration may include any combination of classifiers, such as classes, interfaces, components, and nodes. Within a collaboration, these classifiers may be organized using all the usual UML relationship, including associations, generalizations, and dependencies. In fact, the structural aspects of a collaboration may use the full range of the UML's structural modeling facilities.

A collaboration does not own any of its structural elements. Rather, a collaboration simply references or uses the classes, interfaces, components, nodes, and other structural elements that are declared elsewhere.

Example : Given a Web-based retail system described by a dozen or use cases (such as Purchase Items, Return Items, and Query order), each use case will be realized by a single collaboration. In addition, each of these collaborations will share some of the same structural elements (such as the classes Customer and Order), but they will be organized in different ways.

We'll also find collaboration as deeper inside the system, which represent architecturally significant mechanisms. For example, in this same retail system, we might

have a collaboration called Internode messaging that specifies the details of secure messaging among nodes).

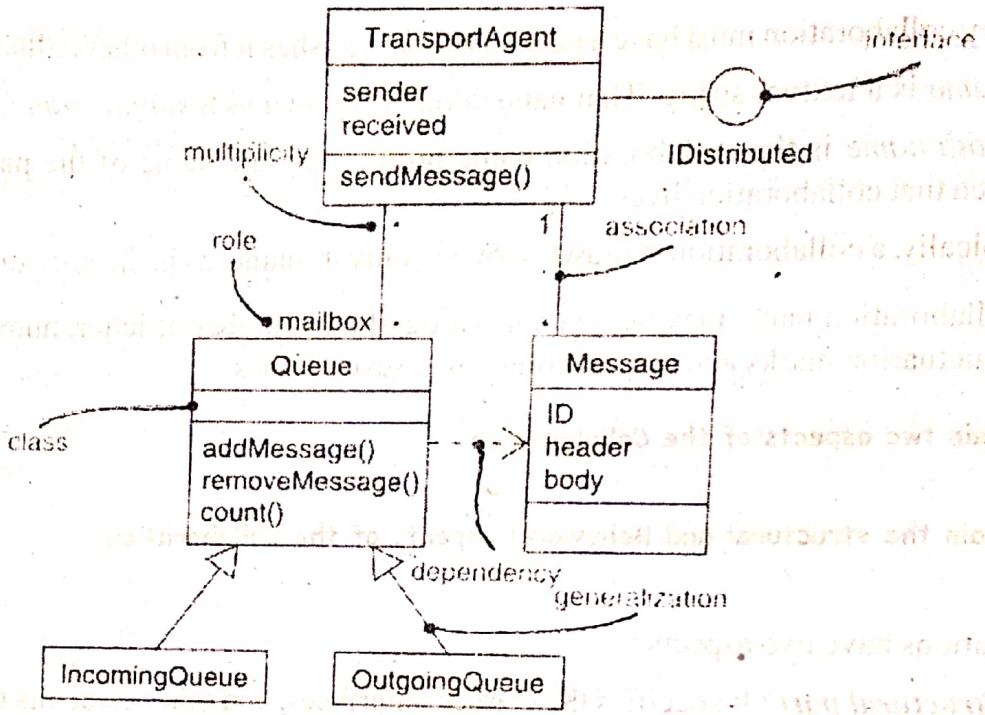


Fig.: Structural Aspects of a Collaboration

Behavioral Aspect

Whereas the structural part of a collaboration is typically rendered using a class diagram, the behavioral part of a collaboration is typically rendered using an interaction diagram. An iteraction diagram specifies an interaction that represents a behavior comprised of a set of messages that are exchanged among a set of objects within a context to accomplish a specific purpose. An interaction's context is provided by its enclosing collaboration, which establishes the classes, interfaces, components, nodes, and other structural elements whose instances may participate in that interaction.

The behavioral part of a collaboration may be specified by one or more interaction diagrams. If we want to empathize the time ordering of messages, use a sequence diagram. If we want to emphasize the structural relationships among these objects as they collaborate, use a collaboration diagram, either diagram is appropriate because, for most purposes, they are semantically equivalent.

Example : This means that when we model a society of classes by naming their interaction as a collaboration, we can zoom inside that collaboration to expose the details of their behavior. For example, zooming inside the collaboration named Internode messaging might reveal the interaction diagram shown in figure above.

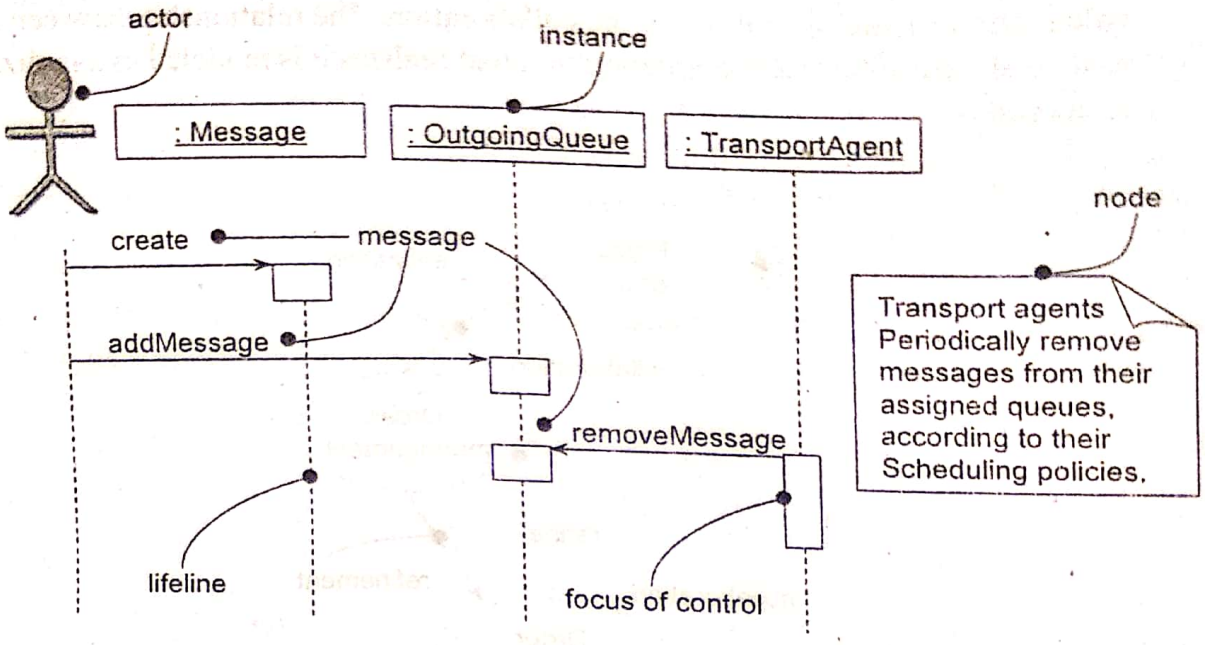


Fig.: Behavioral aspects of a collaboration

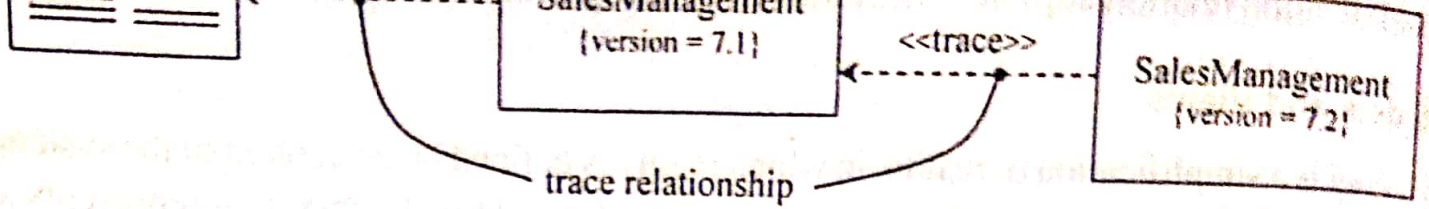


Fig. : Trace Relationships

33. What are the steps involved in modeling the architecture of a system ?

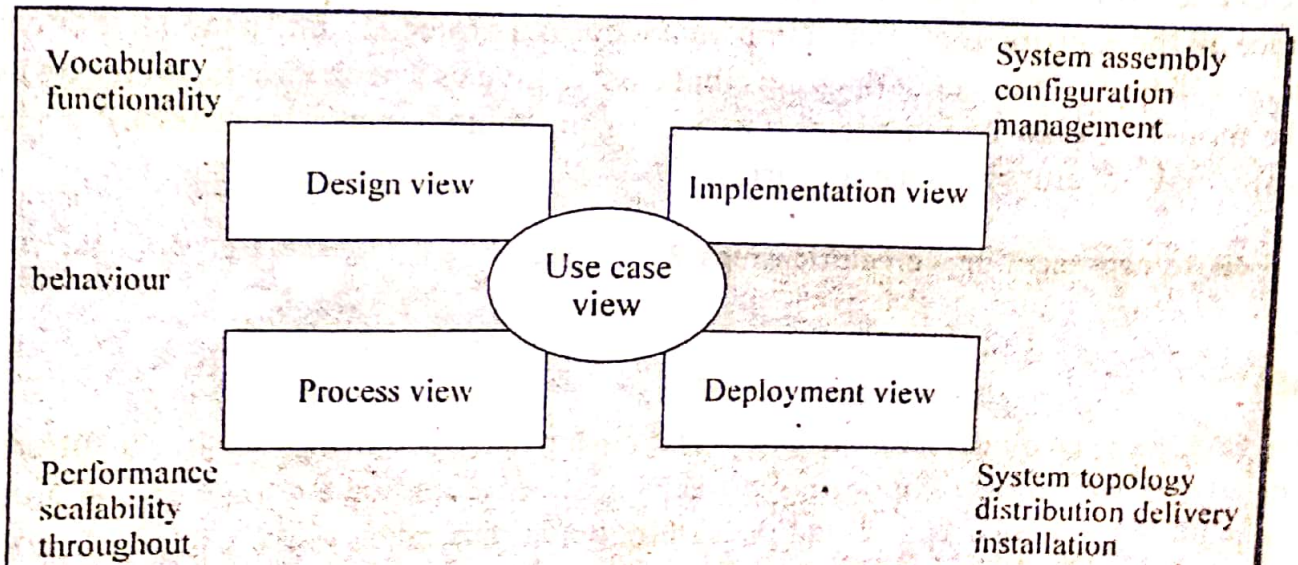
Ans :

Modeling the Architecture of a System

When we model a system's architecture, we capture decisions about the system's requirements, its logical elements and its physical elements. We'll also model both structural and behavioral aspects of the systems and the patterns that shape these views. Finally, we'll want to focus on the seams between subsystems and the tracing from requirements to deployment.

To model the architecture of a system.

1) Identify the views that we use to represent our architecture. Most often we'll want to include a use case view, a design view, a process view, a implementation view, and a deployment view.



3) As necessary, decompose the system into its elementary subsystems.

The following activities apply to the system, as well as to its subsystems.

1) Specify a use case view of the system, encompassing the use cases that describe the behavior of the system as seen by its end users, analysts, and testers. Apply use case diagrams to model static aspects, and interaction diagrams, statechart diagrams, and activity diagrams to model the dynamic aspects.
2) Specify a design view of the system, encompassing the classes, interfaces, and collaborations that form the vocabulary of the problem and its solution. Apply class diagrams and object diagrams to model static aspects, and iteration diagrams, statechart diagrams, and activity diagrams to model the dynamic aspects.
3) Specify a process view of the system, encompassing the threads and processes that form the system's concurrency and synchronization mechanisms.
4) Specify an implementation view of the system, encompassing the components that are used to assemble and release the physical system. Apply component diagrams to model static aspects, and iteration diagrams, statechart diagrams, and activity diagrams to model the dynamic aspects.
5) Specify a deployment view of the system, encompassing the nodes that form the system's hardware topology on which the system executes.
6) Model the architectural patterns and design patterns that shape each of these models using collaborations.

34. What are the steps required to model the system or a sub system ?

Ans :

Modeling Systems of Systems

A system at one level of abstraction will look a subsystem of a higher level of abstraction. Similarly, a subsystem at one level of abstraction will look like a full-fledged system from the perspective of the team responsible for creating it. The development of a subsystem looks just like the development of a system.

To model a system or a subsystem :

1) Identify major functional parts of the system that may be developed released, and deployed somewhat independently. Technical, political, legacy, and legal issues will often shape how we draw the lines around each subsystem.
2) For each subsystem, specify its context, just as we do for the system as a whole; the actors that surround a subsystem encompass all its neighboring subsystems, so they must all be designed to collaborate.
3) For each subsystem, model its architecture just as we do for the system as a whole.