Object Oriented Modeling and Design Patterns.

17MCA51

1 a. **What is Object Orientation? Explain briefly the stages involved in Object Oriented methodology.**

The process consists of building a model of an Application and then adding details to that during design. The same seamless notation is used from analysis to design to implement notation, so that information added in one stage of development need not to be lost or translated for the next stage.

The methodology has the following stages.

- System conception
- Analysis
- System design
- Class design
- Implementation
  - System conception: Begins with business analysts or users conceiving an application and tentative requirements.
  - Analysis:The analyst scrutinizes and rigorously restate the requirements from the system conception by constructing models.The analysis model is a concise,precise abstraction of what the desired system must do,not how it will done.The analysis model should not contain implementation decisions.
    The analysis model had two parts:
    1. Domain model: a description of the real world objects reflected within the system.
    2. *Application model:* a description of the parts of the application system itself that are visible to the user.
  - System Design:The development teamdevise a high- lavel strategy the system architecture for solving the application problem. They also establish policies that will serve as a default for the subsequent, more detailed portion of design .The system designer must decide what performance characteristics to optimize, choose strategy of attacking the problem ,and make tentative resource allocation.
  - Class design: The class designer adds details to the analysis model inaccordance with the system design stratergy. The class designer elaborate both domain and application objects using the same OO concepts and notation, although they exist on different conceptual planes.

- Implementation:Implementation translate the class and relationship developed during class design into a particular programming language, database, or hardware. Programming should be straightforward, because all of the hard decisions should have already been made.

b. Define the following terms with examples.
a) Object b) Class

Object

An object is a real-world element in an object–oriented environment that may have a physical or a conceptual existence. Each object has −

- Identity that distinguishes it from other objects in the system.
- State that determines the characteristic properties of an object as well as the values of the properties that the object holds.
- Behavior that represents externally visible activities performed by an object in terms of changes in its state.

Objects can be modelled according to the needs of the application. An object may have a physical existence, like a customer, a car, etc.; or an intangible conceptual existence, like a project, a process, etc.

Class

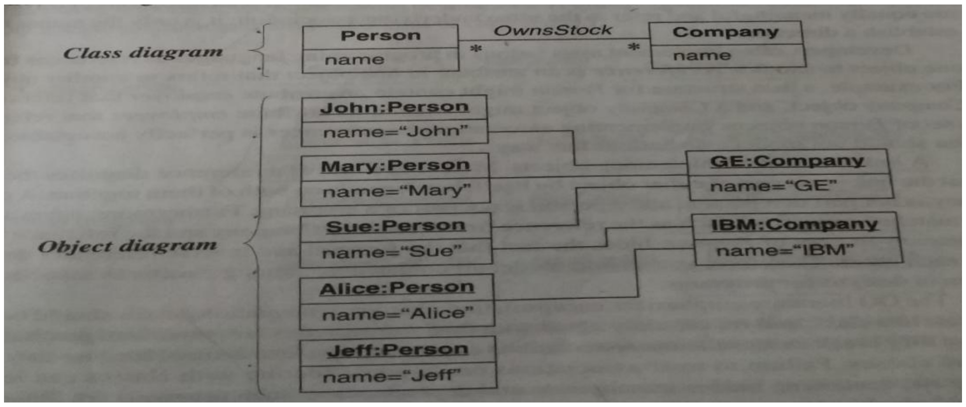
A class represents a collection of objects having same characteristic properties that exhibit common behavior. It gives the blueprint or description of the objects that can be created from it. Creation of an object as a member of a class is called instantiation. Thus, object is an instance of a class.

The constituents of a class are −

- A set of attributes for the objects that are to be instantiated from the class. Generally, different objects of a class have some difference in the values of the attributes. Attributes are often referred as class data.
- A set of operations that portray the behavior of the objects of the class. Operations are also referred as functions or methods.

2 a. **What are links and association? Write and explain UML notations for links and associations with an example.**
A ***link*** is a physical or conceptual connection among objects. For example, Joe Smith *Works-For* Simplex Company. Most links relate two objects, but some links relate three or more objects. Mathematically, we define a link as a tuple – that is, a list of objects. A link is an instance of an association.

An ***association***is a description of a group of links with common structure and common semantics. For example, a person *WorksFor* a company. The links of an association connect objects from the same classes. An association describes a set of potential links in the same way that a class describes a set of potential objects. Links and associations often appear as verbs in problem statements.
The figure below is an excerpt of a model for a financial application. Stock brokerage firms need to perform tasks such as recording ownership of various stocks, tracking dividends, alerting customers to changes in the marker, and computing margin requirements. The top portion of the figure shows a class diagram and the bottom shows an object diagram.



b. Discuss aggregation and generalization with example.

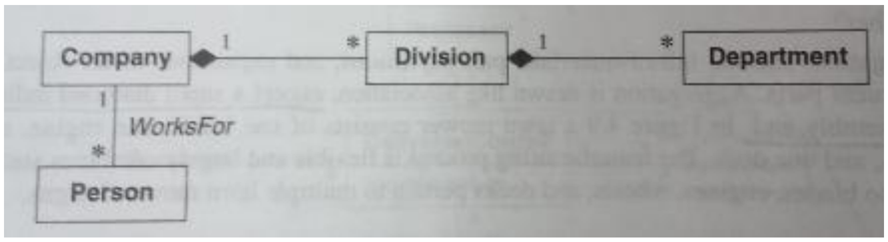
Aggregation: It is a Strong form of Associtation in which an aggregate object is made of constituent parts. Constituent are the part of the aggregate. The aggregate is semantically an extended object that is treated as unit in many operation, although it is physically made of lesser object.

We define an aggregation as relating an assembly class to one constituent part class. An assembly with many kinds of constituent parts corresponds to many aggregation. For example, a Lawnmower consists of blade, an Engine, many wheels, and a Deck. Lawnmower is the assembly and the other parts are constituents. Lawnmower to Blade is one aggregation, Lawnmower to Engine is other aggregation, and so on. We define each individual pairing assembly. This definition emphasizes that aggregation is special form of binary association .

The most significant property of aggregation is transitivity that is, if A is part of B and B is part of C, then A is part of C. Aggregation is also Antisymmetric that is, if A is part of B, then B is not part of a, Many aggregate operations imply transitivity closure and operate an both direct and indirect parts.

Composition: It is a form of aggregation with two additional constraints.A constituent part can belong to at most one assembly.Furthurmore,once a constituent part has been assigned an assembly,it has a coincident lifetime with assembly.thus composition implies ownership of the parts by the whole. This can be convenient for programming:Deletion of an composition is small solid diamond next to assembly class(vs.a small hallow diamond for the general form of aggregation)

In fig 4.10 a Company consists of divisions, which in turn consist of departments:a company is indirectly a composition of departments .A company is not a compositin of its employee,since company and the person are independent object of equal stature.



3 a. **What do you mean by state and events? Discuss the state diagram for a telephone line system with activities.**

An Event is an Occurance at a Point in time,such as user depress left button or flight123 departs from Chicago.Events often correspond to verbs in the past tense(power turned on,alarm set) or to the onset of some condition(paper tray becomes empty,temerature becomes loer than freezing).by defination,an event happens instaneouslyith regard to thge time scale of an application.ofcoure,nothing is really instaneous;anevenyt is simply an occurance than the application considers atomic and fleeting.the time at vhich an evnet occurs is an implicit of the event.

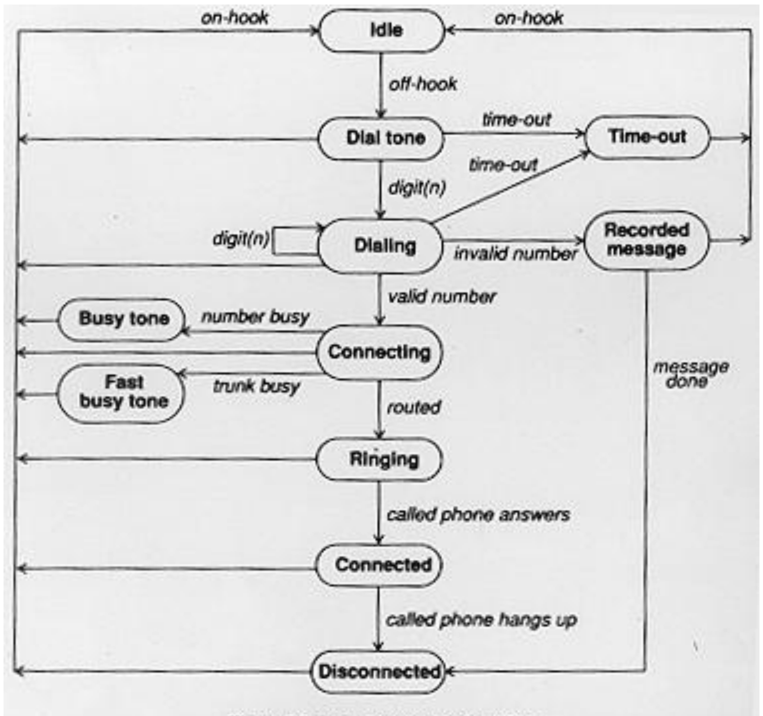
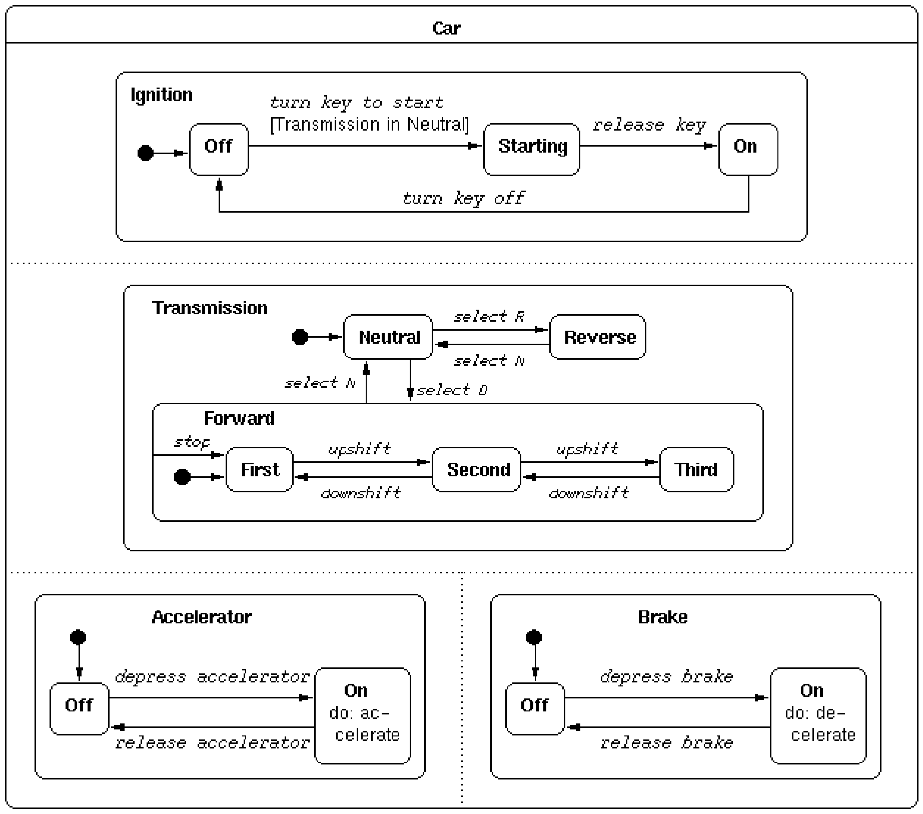
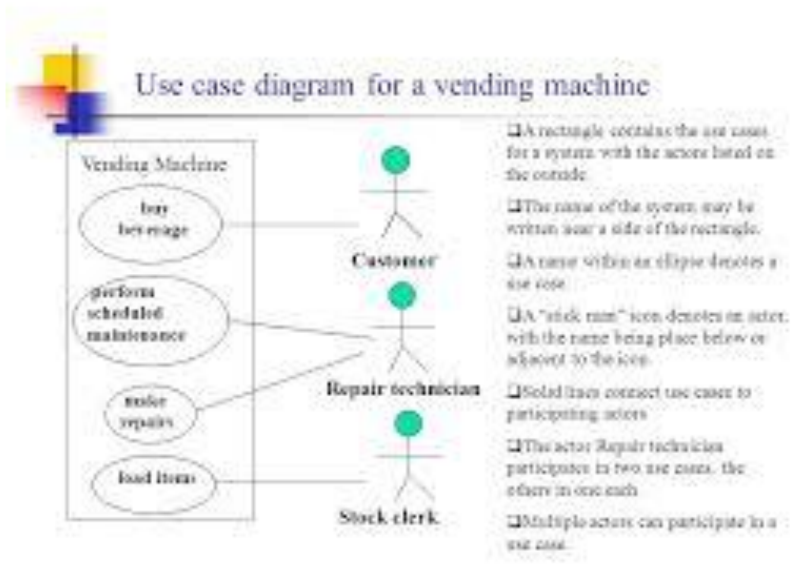


Figure 5.5 State diagram for phone line

b. Discuss aggregation concurrency with state of a car as an aggregation of parts.



4 a. Discuss the use case diagram for vending machine. What are the guidelines needed to be followed for use-case models?



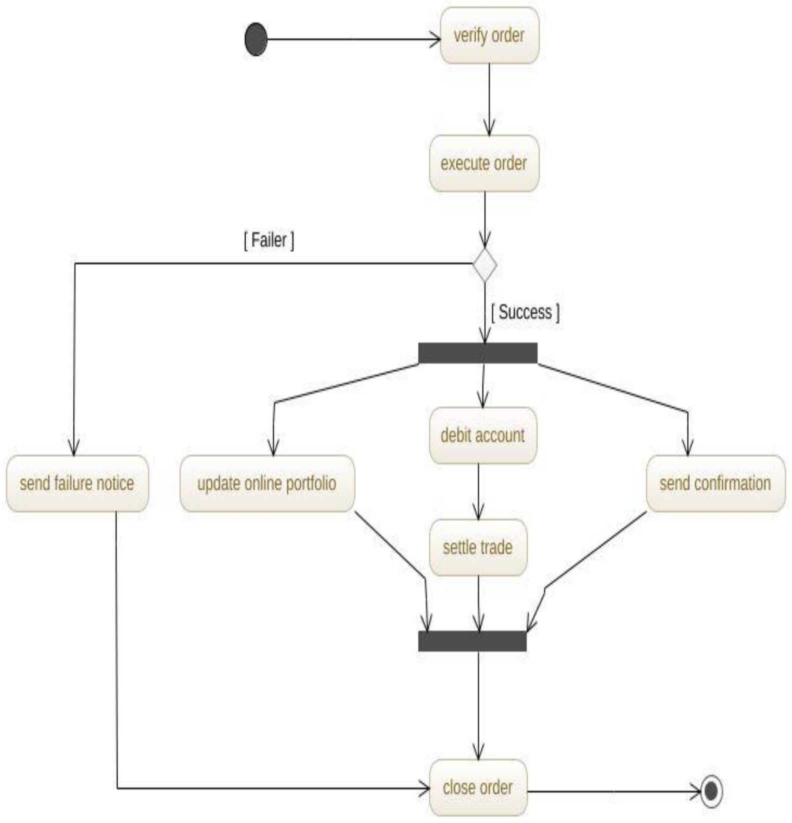
Use case diagram for a vending machine

- ❑A rectangle contains the use cases for a system with the actors listed on the outside
- ❑The name of the system may be written near a side of the rectangle.
- ❑A name within an ellipse denotes a use case
- ❑A "stick man" icon denotes an actor, with the name being place below or adjacent to the icon.
- ❑Solid lines connect use cases to participating actors
- ❑The actor Repair technician participates in two use cases, the others in one each
- ❑Multiple actors can participate in a use case.

Guidelines for use case models

Determine the system boundary
Ensure that actors are focused
Each use case must provide value to user
Relate use case and actor
Remember that use cases are informal
Use cases can be structure

b. What are sequence models? Discuss the sequence diagram for a stock purchase.



5 a. **Discuss the different stages in software development process.**

Software development has a sequence of well defined stages each with a distinct purpose input and output

- **System Conception**: Conceive an application and formulate tentative requirements.
- **Analysis**:Deeply understand the requirements by constructing models.The of analysis is to specify what needs to be done,not how it is done.You must understand a problem before attempting a solution.
- **System design**:Device a high level strategy—the architecture—for solving the application problem.Establish policies to guide the subsequent class design.
- Class design:Argument and adjust the real-world models from analysis so that they are menable to computer implementation.Determine algorithms for realizing the operations.
- **Implementation**:Translate the design into programming code and database structures.
- Testing:Ensure that the application is suitable for actual use and that it truly satisfies the requirements.
- **Training**:Help users master the new application.
- **Deployment**:Place the application in the field and gracefully cut over from legacy applications.
- **Maintenance:**Preserve the long-term viability of the application.

b**. List and explain questions that must be answered by a good system concept.**

- You should clearly understand who is the stake holders of the new system
- Two important kinds of stakeholders are
  1.financial Sponsers
  2.End Users
- Financial Sponsers: They are important because they are paying for the new system, and they expect the project to be on schedule and within budget.
- End users:They will ultimately detmine the success of the new system by an increase in their productivity or effectiveness.

I. What problems will it solve?
- You must clearly bound to the size of the effort and estaqblish its scope
- Determine features of the system by consulting various users of the system

II. Where it will be used?
- You should where the new system is used and determine if the system is mission-critical software for the organization, experimental software,or a new capability that you can deploy without disrupting the workflow.

III. When it is needed?
- Two aspects of time are important.
- First the feasible time, the time in which the system can be developed within the constraint of cost and available resources
- Second the required time, when the system is needed to meet business goals

IV. Why it is needed?
- Prepare a business case if it is not prepared for the new system.
- It contains the financial justifications for the new system,including the cost, tangible benefits, intangible benefits ,risks and alternatives.

V. How will it work?
- You should brainstorm about the feasibility of the problem.
- For large system you should consider the merits of different architectures.
- Here the purpose is not to choose the solution, but to increase the confidence that the problem can be solved reasonably.

A good system concept must answer the following questions:

VI. Who is the application for?

6.a. **List the steps to construct a domain class model for an ATM bank system, prepare data dictionary for all modeling elements.**

The steps to be performed to construct a domain class model:

1. Find Classes.
2. Prepare a data dictionary.
3. Find associations.
4. Find attributes of objects and links.
5. Organize and simplify classes using inheritance.
6. Verify that access paths exist for likely queries.
7.Iterate and refine the model.
8.Reconsider the level of abstraction .
9.Group classes into packages

b**. How to construct an application class model?**

1. Specify user interfaces
2. Define boundary classes
3. Determine controllers
4. Check against the interaction mode

1. Specify user interfaces User interface a. Is an object or group of objects b. Provide user a way to access system's

i. domain objects,
ii. ii. commands, and
iii. iii. Application options. Try to determine the commands that the user can perform. A command is a large-scale request for a service, c. E.g.
iv. 1. Make a flight reservation 2. Find matches for a phrase in a database Decoupling application logic from the user interface. ATM example - The details are not important at this point.

2. Defining Boundary Classes

• A boundary class – Is an area for communications between a system and external source. – Converts information for transmission to and from the internal system.

• ATM example

• CashCardBoundary

• AccountBoundary – Between the ATM and the consortium ATM

3. Determining Controllers

• Controller is an active object that manages control within an application.

Controller – Receives signals from the outside world or – Receives signals from objects within the system, – Reacts to them, – Invokes operation on the objects in the system, and – Sends signals to the outside world. ATM Example

• There are two controllers – The outer loop verifies customers and accounts. – The inner loop services transactions.

Analysis Stereotypes

• <> classes in general are used to model interaction between the system and its actors.

• <> classes in general are used to model information that is long-lived and often persistent.

• <> classes are generally used to represent coordination, sequencing, transactions, and control of other objects. And it is often used to encapsulate control related to a specific use case.

4. Checking Against the Interaction Model

• Go over the use cases and think about how they would work.

• When the domain and application class models are in place, you should be able to simulate a use case with the classes.

7 a. Discuss how to construct applications state model for ATM.

The following steps are performed in constructing a domain state model:
1. Identify domain classes with states
2. Find states.
3. Find events.
4. Build state diagrams.
5. Evaluate state diagrams.

1. Identifying classes with states:
Examine the list of domain classes for those that have a distinct life cycle. Look for classes that can be characterized by a progressive history or that exhibit cyclic behavior identify the significant states in the life cycle of an object.
ATM example. Account is an important business concept, and the appropriate behavior for an ATM depends on the state of an account. The life cycle for account is a mix of progressive and cycling to and

2. Finding states: List the states for each class. Characterize the objects in each class-the attribute values that an object may have, the associations that it may participate in and their multiplicities. Give each state a meaningful name. Don’t focus on fine distinction among states, particularly quantitative differences. ATM example. Here are some states for an Account: Normal(ready for normal access),Closed(closed by the customer but still on file in the bank records), Overdrawn(customer

withdrawals exceed the balance in the account),and Suspended(access to the account in blocked for some reason).

3. Finding events: Once you have a preliminary set of states, find the events that cause transitions among states. Think about the stimuli that cause a state to change. In many cases you can regard an event as completing a do-activity. In cases of completing a do-activity, other possibilities are often possible and may be added in the future-for example conditionally accept with revisions.

b. Explain the guidelines for activity models.

The flow of events of a use case can be described graphically with the help of an activity diagram. Such a diagram shows:

- Activity states, which represent the performance of an activity or step within the flow of events.
- Transitions that show what activity state follows after another. This type of transition is sometimes referred to as a completion transition, since it differs from a transition in that it does not require an explicit trigger event, it is triggered by the completion of the activity the activity state represents.
- Decisions for which a set of guard conditions are defined. These guard conditions control which transition (of a set of alternative transitions) follows once the activity has been completed. Decisions and guard conditions allow you to show alternative threads in the flow of events of a use case.
- Synchronization bars which you can use to show parallel subflows. Synchronization bars allow you to show concurrent threads in the flow of events of a use case.

8 .Explain the concept of Refactoring and Design Optimization.

The analysis model captures the logical information about the system, while the design model adds details to support efficient information access. Before a design is implemented, it should be optimized to make the implementation more efficient. The aim of optimization is to minimize the cost in terms of time, space and other metrics.

However, design optimization should not be excess, as ease of implementation, maintainability and extensibility are also important concerns. It is often seen that a perfectly optimized design is more efficient but less readable and reusable. So, the designer must strike a balance between the two.

The various things that may be done for design optimization are-

- Add redundant associations
- Omit non-usable associations
- Optimization of algorithms
- Save derived attributes to avoid re-computation of complex expressions

Addition of Redundant Associations

During design optimization, it is checked if deriving new associations can reduce access costs. Though these redundant associations may not add any information, they may increase the efficiency of the overall model.

Omission of Non-Usable Associations

Presence of too many associations may render a system indecipherable and hence reduce the overall efficiency of the system. So, during optimization, all non-usable associations are removed.

Optimization of Algorithms

In object-oriented systems, optimization of data structure and algorithms are done in a collaborative manner. Once the class design is in place, the operations and the algorithms need to be optimized.

Optimization of algorithms is obtained by –

1. Rearrangement of the order of computational tasks
2. Reversal of execution order of loops from that laid down in the functional model
3. Removal of dead paths within the algorithm

Saving and Storing of Derived Attributes

Derived attributes are those attributes whose values are computed as function of other attributes (base attributes). Re-computation of the values of derived attributes every time they are needed is a time-consuming procedure. To avoid this, the values can be computed and stored in their forms.

However, this may pose update anomalies, i.e., a change in the values of base attributes with no corresponding change in the values of the derived attributes. To avoid this, the following steps are taken –

1. With each update of the base attribute value, the derived attribute is also recomputed.
2. All the derived attributes are re-computed and updated periodically in a group rather than after each update.

9 a. What are design patterns? Discuss structural, creational and behavioral design patterns.

A *design pattern* provides a scheme for refining the subsystems or components of a software system, or the relationships between them. It describes a commonly-recurring structure of communicating components that solves a general design problem within a particular context [GHJV95].

Design patterns are medium-scale patterns. They are smaller in scale than architectural patterns, but tend to be independent of a particular programming language or programming paradigm. The application of a design pattern has no effect on the fundamental structure of a software system, but may have a strong influence on the architecture of a subsystem.

Many design patterns provide structures for decomposing more complex services or components. Others address the effective cooperation between them, such as the following pattern:

Name Observer [GHJV95] or Publisher-Subscriber (339)

Context A component uses data or information provided by another component.

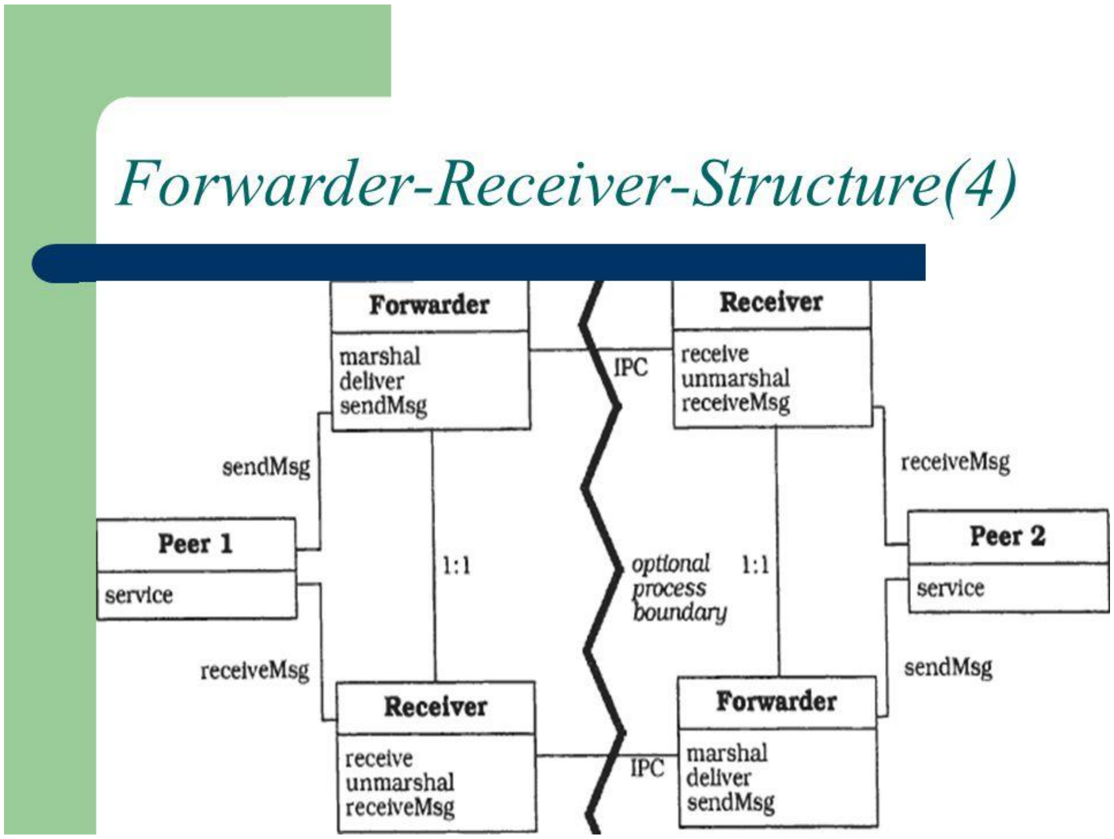
Problem Changing the internal state of a component may introduce inconsistencies between cooperating components. To restore consistency, we need a mechanism for exchanging data or state information between such components.

Two *forces* are associated with this problem:

- The components should be loosely coupled—the information provider should not depend on details of its collaborators.
- The components that depend on the information provider are not known a priori.

Solution Implement a change-propagation mechanism between the information provider—the *subject*—and the components dependent on it—the *observers*. Observers can dynamically register or unregister with this mechanism. Whenever the subject changes its state, it starts the change-propagation mechanism to restore consistency with all registered observers. Changes are propagated by invoking a special update

b. Describe forwarder-receiver design pattern.



10a. Explain the concept of whole-part design pattern with a suitable example.

The Whole- Part design pattern helps with the aggregation of components that together form a semantic unit. An aggregate component, the Whole, encapsulates its constituent components, the parts, organizes their collaboration, and provides a common interface to its functionality, Direct access to the parts is not possible.

Example:

A computer-aided design[CAD] system for 2-D and 3-D modeling allows engineers to design graphical objects interactively. In such systems most graphical objects are modeled as compositions of other objects. For example, a car object aggregates several smaller objects such as wheels and windows, which themselves may be composed of even smaller objects such as circles and polygons. It is the responsibility of the car as a whole, such as rotating or drawing.

Context:

Implementing aggregate objects.

Problem:

In almost every software system objects that are composed of other objects exist. For example, Consider a molecule object in a chemical simulation system-it can be implemented as a graph of separate atom objects. Such aggregate objects do not represent loosely-coupled sets of components. Instead, they form units they are more than just a mere collection of their parts. The molecules example illustrates the typical case in which aggregates reveal behavior that is not obvious or visible from their individual parts- the combination of parts makes new behavior emerge. Such behavior is called emergent behavior.

We need to balance the following forces when modeling such structures:

- A Complex object should either be decomposed into smaller objects, or composed of existing objects, to support reusability, changeability and the recombination of the constituent objects in other types of aggregate.
- Clients should see the aggregate object as an atomic object that does not allow any direct access to its constituent parts.

Solution:

Introduce a component that encapsulates smaller objects, and prevents clients from accessing these constituent parts directly. Define an interface for the aggregate that is the only means of access to the functionality of the encapsulated objects, allowing the aggregate to appear as a semantic unit.

b. Discuss the concept of architectural pattern.

aspects such as transaction policies in business applications, or call routing in telecommunication.

To refine our classification, we group patterns into three categories:

- Architectural patterns
- Design patterns
- Idioms

Each category consists of patterns having a similar range of scale or abstraction.

Architectural Patterns

Viable software architectures are built according to some overall structuring principle. We describe these principles with *architectural* patterns.

An *architectural pattern* expresses a fundamental structural organization schema for software systems. It provides a set of predefined subsystems, specifies their responsibilities, and includes rules and guidelines for organizing the relationships between them.

Architectural patterns are templates for concrete software architectures. They specify the system-wide structural properties of an application, and have an impact on the architecture of its subsystems. The selection of an architectural pattern is therefore a fundamental design decision when developing a software system.

The Model-View-Controller pattern from the beginning of this chapter is one of the best-known examples of an architectural pattern. It provides a structure for interactive software systems.

Design Patterns

The subsystems of a software architecture, as well as the relationships between them, usually consist of several smaller architectural units. We describe these using *design* patterns.