Chapter – 5

Object Oriented Analysis and Design for a system

Objectives

At the end of this Chapter, the students will be able to:

- Use object, class, and inheritance etc to design the system.
- Application of object oriented methods to a system development.
- Recognize in what circumstances an object-oriented approach is applicable.
- Demonstrate how each principle and concept is applied for a system.

Introduction

This chapter introduces the system development life cycle with major focus on analysis and design of system using object-oriented techniques, which view a system as a collection of self-contained objects that have both data and processes. This change has been accelerated through the creation of the Unified Modeling Language (UML). UML provides a common vocabulary of object-oriented terms and diagramming techniques that is rich enough to model any systems development project from analysis through implementation.

5.1 System Design

System design is a high level strategy for solving the problem and building a solution, system design includes decisions about the organization of the system into subsystems, the allocation of subsystems to hardware and software components.

The overall organization of a system is called the system architecture. There are a number of common architectural styles, each of which is suitable for certain kinds of applications. One way to characterize an application is by the relative importance of its object dynamic and functional models. Different architectures place differing emphasis on the three models.

5.2 Object Design

Overview of Object Design

During object design the designer carries out the strategy chosen during system design and fleshes out the details. There is a shift in emphasis from application domain concept towards computer concepts. Optimization of the design should not be carried to excess, as case of implementation, maintenance, and extensibility are also important concerns.

Working from Analysis and Architecture

The information in the analysis object model must be present in the design in some form.

The best and simple approach is to carry the classes from analysis to design. Implementation of making decisions will be in detail. New redundant classes are added for efficiency.

The operations which the system must implement are given by the functional model. Here selecting algorithms for operations and breaking operations are done:

Decomposition is an iterative process and must be repeated at successively lower levels of abstraction.

- ➢ The algorithms and decomposition must be chosen to optimize important implementation measures.
- ➢ The implementation measures may include implementation understandability and performance.
- ➢ The dynamic model describes how the system responds to external events. The control structure for a program is primarily derived from the dynamic model.
- ➢ Explicit or implicit recognition of flow of control of data within the system should be done.
- ➢ The decision to map events depends on the architecture. Object oriented design is primarily a process of refinement.

Steps of Object Design

Object –oriented design is an iterative process. It may be necessary to revise the relationships between objects. New operations and attributes must be added to classes in the object model and new class will be identified.

The following steps must be performed during the phase of object design:

- ➢ Combining the three models to obtain operation on classes.
- ➢ To implement operations design algorithms are used
- ➢ Optimize access paths to data
- ➢ Implement control for external interactions
- ➢ Adjust class structure to increase inheritance
- ➢ Design associations
- ➢ Determine object representation
- ➢ Package classes and associations into modules

5.3 Design Algorithms for Functional model

Each operation specified in the functional model must be formulated as an algorithm.

- ➢ An algorithm may be subdivided into calls on similar operations

There are certain rules which the algorithm designer must keep in mind:

- ➢ Choose algorithms that minimize cost of implementing operations
- ➢ Appropriate selection of data structures
- ➢ Internal classes and operations are to be defined
- ➢ Assign responsibility for operations to appropriate classes

5.4 Design Optimization

Basic design model uses the analysis model as the framework for implementation

- ➢ The analysis model captures the logical information about the system
- ➢ Insufficient model can be optimized to make the implementation more efficient
- ➢ Optimized system is more obscure and less likely to be reusable in another context.

During design optimization there are certain necessary things that has to be carried out.

- ➢ Add redundant associations to minimize access cost and maximize convenience
- ➢ For greater efficiency the computation must be rearranged
- ➢ Save derived attributes to avoid re-computation of complicated expressions

5.4.1 Adding Redundant Association for Efficient Access

During design we evaluate structure of the object model for implementation. The association that were useful during analysis may not form the most efficient network when the access paths and relative frequencies of different kinds of access are considered

To use the paths in the association of network can be analyzed as follows

- ➢ Each operation must be examined and find what are the associations in which it must traverse to obtain its information
- ➢ That association that are traversed in both the directions must be noted
- ➢ The associations that are traversed in single direction must be noted
- ➢ This can later be implemented effectively by pointers

5.4.2 State as location within a program

A traditional approach to represent control within a program is:

- ➢ The location of control within a program implicitly defines the program state

- Each state transition corresponds to an input statement
- After input is read, the program branches depending on the input event received
- Highly nested procedural code, must be passed to low level procedures & procedure calls until some procedure is prepared to handle them
- The lack of modularity is the biggest drawback of this approach

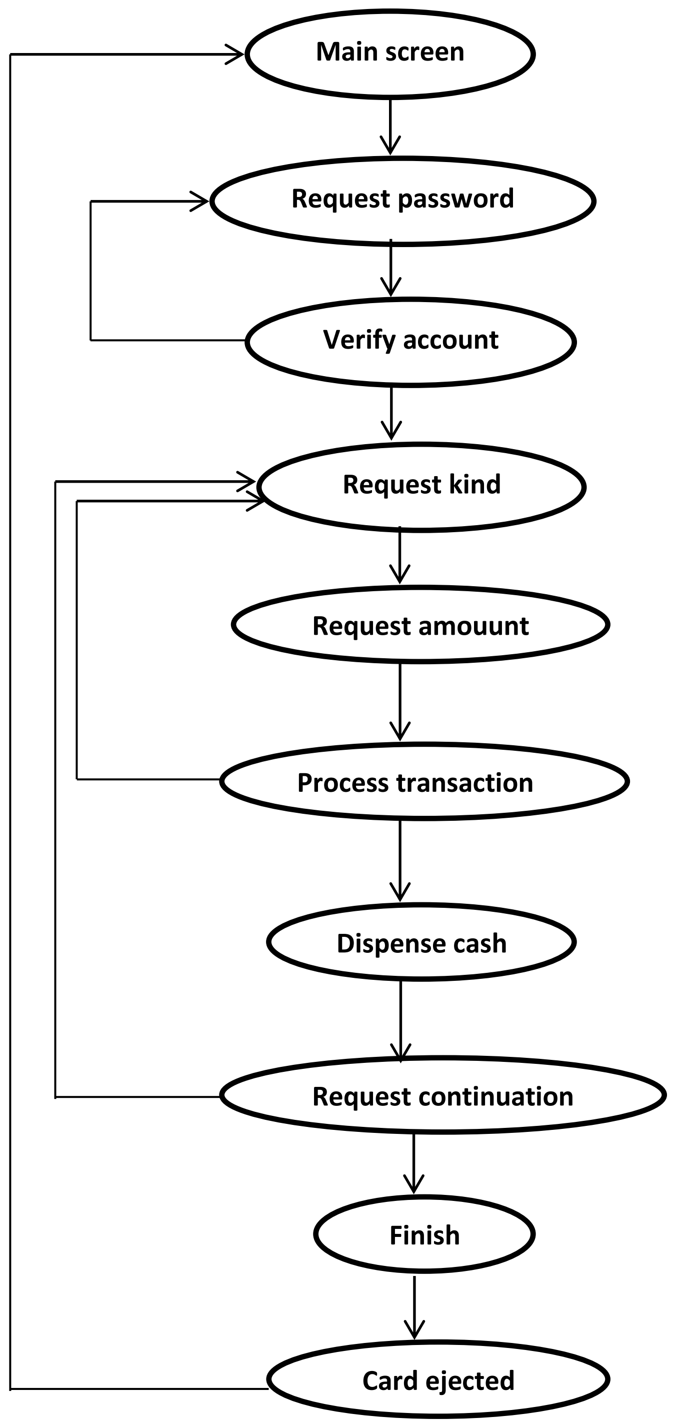
One technique of converting a state diagram to code is as follows;

1. Identify the main control path
2. Identify alternate paths that branch off the main path and rejoin it later
3. Identify backward paths that branch off the main loop and rejoin in later
4. The states and transitions that remain correspond to the exceptional conditions

Below given approach could be applied to the state model for the ATM class introduction

This gives the state model, which identify the main path of control, which corresponds to the reading of a card, querying the user for transaction information, processing the transaction, printing a receipt and electing the card.

Psuedocode



```
do forever
     display main screen
       read card
       repeat
                  ask for password
                        read password
                        verify account
until account verification is OK
repeat
 repeat
        ask for kind of transaction
        read kind
        ask for amount
        road amount
        start transaction
        wait for it to complete
 until transaction is OK
 dispense cash
 wait for customer to take it
  ask whether to continue
  until user asks to terminate
  eject card
  wait for customer to take card
```

ATM Control

Input events within the single- thread program are coded as blocking I/O reads, that is, I/O statements that wait for input

The operating system is responsible for catching interrupts and queuing them up for ordinary programs

5.4.3 Rearranging Classes and Operations

The same operation is defined across several classes and can easily be inherited from a common ancestor, but more often operation in different classes may be similar but not identical. Before inheritance can be used, each operation must have the same interface and the same semantics. The following kind of adjustment s can be used in the increase of the change of inheritance.

- Some operations may have fewer arguments than others
- Some operations may have fewer arguments because they are special cases of more general arguments
- Similar attributes in different classes may have different names.
- An operation may be defined on several different classes in a group but be undefined on the other classes.

5.4.4 Abstracting Out Common Behavior

- New classes and operations are often added during design. If a set of operations and or attributes seem to be repeated in two classes then they may be specified variations of the same thing at higher level of abstraction
- When common behavior has been recognized, a common super class can be created that implements the shared features
- This transformation of the object model is called abstracting out a common super class
- Sometimes it is worthwhile to abstract out a super class even when there is only one sub class in project that inherits from it
- Abstract super class have benefits other than sharing and reuse
- Splitting of a class into two classes that separate the specific activities from the more general aspects is a form of modularity
- There is a subtle but important way that abstract super-class improve the configuration management aspect of software maintenance and distribution

5.4.5 Use Delegation to Share Implementation

Inheritance is a mechanism for implementing generalization, in which the behavior of a super class is shared by all its sub classes. Sharing of behavior is justifiable only when a true generalization relationship occurs.

Operations of the sub class that override the corresponding operation of the super class have a responsibility to provide the same services provided by the super class.

When class B inherits the specification of class A, we can assume that every instance of class B is an instance of class A because it behaves the same.

- Generally, programmers use inheritance as an implementation to technique with no intention of guaranteeing the same behavior
- The designer is then tempted to inherit from the existing class to achieve part of the implementation of the new class.

5.5 Design of Associations

Associations are the "glue" of any object model, providing access paths between objects. Associations are conceptual entities useful for modeling and analysis. During the object design phase formulation of a strategy for implementing the associations in the object model must be done.

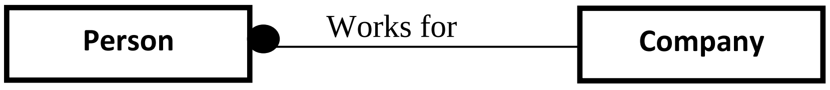
5.5.1 Analyzing Association Traversal

The assumption until now is that the associations are inherently bidirectional, which is certainly true in an abstract sense. But if some associations in application are only traversed in one direction, their implementation can be simplified.

For prototype work, we always use bidirectional association so that we can add new behavior and expand or modify the application rapidly.

5.5.2 One – way Associations

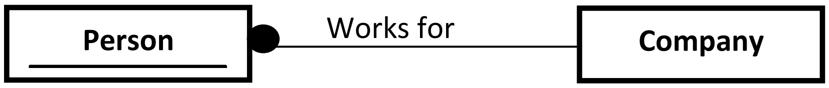
- if an association is only traversed in one direction it may be implemented as a pointer
- if the multiplicity is "one" then it is a simple pointer, if the multiplicity is "many" then it is set of pointers. If the "many" end is ordered, then a list can be used instead of set.
- Qualified associations with multiplicity "many" are rare, but they can be implemented as a directory of set of objects.



5.5.3 Two – Way Associations

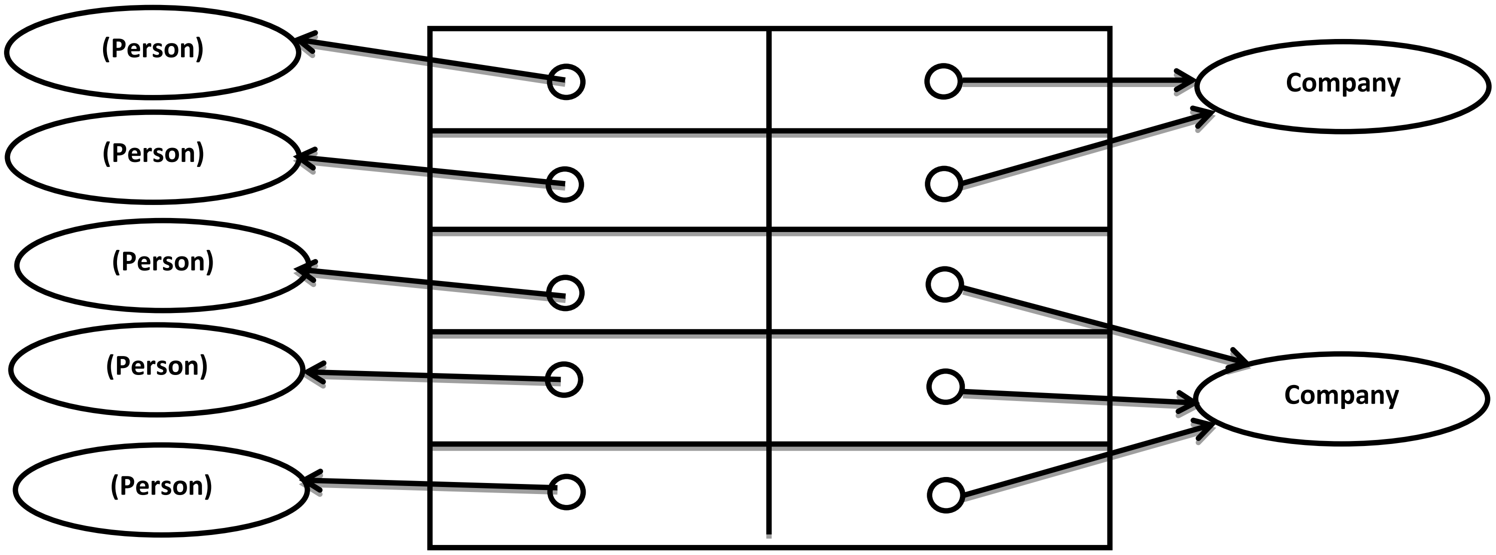
Many associations are traversed in both directions, although not usually with equal frequency. There are three approaches to their implementation.

- Implementation as an attribute in one direction only and perform a search when a backward traversal is required.



Implementation of two –way association using pointers

- Implement as attributes in both directions, using the technique as in the above figure.
- Implement as a distinct association object, independent of either class. An associated object is a set of pairs of associated objects stored in single variable size object.



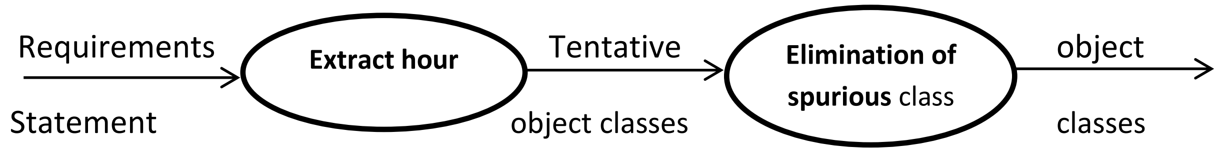
Implementation of two-way association using pointers

5.6 Object Modeling

The first step in analyzing the requirements is to construct an object model. The object model shows the static data structure of the real world system and organizes it into workable prices. The object model describes real- world object classes and their relationship to each other. Most crucial one is the top level organization of the system into classes connected by associations; lower level partitions within classes (generalization) are less critical. The object model precedes the dynamic model and functional model because static structure is usually better defined, less dependent on application details, more stable as the solution evolves and easier for humans to understand.

5.6.1 Identifying Object Classes

The first step in constructing an object model is to identify relevant object classes from the application domain. Objects include physical entities such as house employees and machines as well as concepts such as trajectories, seating assignments and payment schedules. Let us consider an example of a reservation system to sell tickets to performance at various theatres. Tentative classes would be reservation system, ticket, performance and theatre.



5.6.1.1 Keeping the Right Classes

While identifying the object classes it is very important identify the right class and discard the unnecessary and incorrect classes. Like (1)Redundant classes (2) irrelevant classes (3) vague classes (4)attributes (5)operations (6)roles (7) implementation constructs.

5.6.2 Preparing a Data Dictionary

Isolated words have too many interpretation, so prepare a data dictionary for all modeling entities write a paragraph precisely describing each object class describe the scope of the class within the current program including any assumptions or restrictions on its membership or use. The data dictionary also describes associations, attributes and operations.

5.6.3 Identifying Association

Identifying association between classes: Any dependency between 2 or more classes is an association. A reference from one class to another is an association.

Associations show dependencies between classes at the same level of abstraction as the classes themselves, while object –valued attributes hide dependencies and obscure their two way nature. Association can be implemented in various ways, but such implementation, decisions should be kept out of the analysis model to preserve design freedom

5.6.3.1 Keeping the Right Association

While identifying the right association we should discard the unnecessary and incorrect association.

1. Association between eliminated classes

2. Irrelevant or implementation associations
3. Actions
4. Ternary association
5. Derived association
6. is named association
7. Role names
8. Qualified associations
9. Multiplicity
10. Missing association

5.6.4 Identifying Attributes

Identifying object attributes: attributes are properties of individual objects as name, weight, velocity or color. Attributes should not be objects use an association to show any relationship between two objects.

Attributes usually corresponds to nouns followed by possessive phrases such as " the color of the car" or "the position of the cursor" etc.

5.6.4.1 Keeping the Right Attributes

Eliminate unnecessary and incorrect attributes

1. Objects
2. Qualifies
3. Names
4. Identifiers
5. Link attributes
6. Internal values
7. Fine details
8. Discordant attributes

5.6.5 Refining with Inheritance

We need to organize classes by using inheritance to share common structure. Inheritance can be added in two directions by generalizing common aspects of existing classes into a super-class (bottom up) or by refining existing classes into specialized sub classes (top down)

You can discover inheritance from the bottom up by searching for classes with similar attributes, association or operations

Top down specialization are often apparent from the application domain.

Multiple inheritance may be used to increase sharing but only if necessary because it increase both conceptual and implementation complexity

5.6.6 Testing Access Paths

Trace access paths through the object model diagrams. Something that seems simple in the real world appears complex in the model & it should be made sure that the complexity is not inherent in the real world

5.7 Dynamic modeling

Dynamic model shows the time – dependent behavior of the system and the object s in it. Dynamic modeling analysis begins with the looking for events externally – visible stimuli and responses.

Summarize permissible event sequences for each object with a state diagram

The dynamic model is insignificant for a purely static data repository, such as a database. The dynamic model is important for interactive systems.

First prepare scenarios of typical dialogs. The scenarios may not cover every contingency. Extract events from scenarios and identify events first and then assign each event to its target. Organize the sequences of events into a state diagram. The resulting set of state diagrams constitutes the dynamic model.

5.7.1 Preparing Scenario

These scenarios show the major interactions, external display formats, and information exchanges.

The problem statement describes the full interaction sequence, but most of the time we have to invent the interaction format.

First prepare scenarios for “normal” cases, interactions without any unusual inputs or error conditions. Then consider “special” cases, such as omitted input sequences, maximum and minimum values, and repeated values.

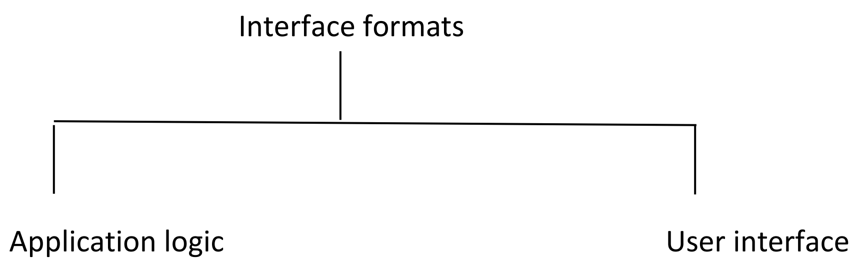
Consider user error cases, including invalid values and failures to respond. For many interactive applications, error handling is the most difficult part of implementation

Finally, consider various other kinds of interaction that can be overlaid on basic interactions such as help requests and status queries

A scenario is a sequence of events. An event occurs whenever information is exchanged between an object in the system and an outside agent, such as user, a sensor, or another task. The information values exchanged are parameters of the event

5.7.2 Interface Formats

Interactions can be separated into two parts: application logic and the user interface.



The analysis should concentrate first on the information flow and control, rather the on the presentation format.

The dynamic model captures the control logic of the application

Application logic can often be simulated with dummy procedures. Decoupling application logic from the user interface allows the interface to be evaluated while the application is under development

5.7.3 Identifying Events

Events include all signals, inputs, decisions, interrupts, transitions, and actions to or from uses or external devices. Internal computation steps are not events, except for decision points that interact with the external world

An action by an object that transmits information is an event

Group all events under single name that have the same effect on flow of control, even if their parameter values differ.

- You must decide when differences in quantitative values are important enough to distinguish
- Allocate each type of event to the object classes that send it and receive it. The event is an output event for the sender and an input event for the receiver. Sometimes an object

sends an event to itself, in which the event is both an output and an input for the same class

- Show each scenario as an event trace, an ordered list of events between different objects assigned to columns in a table
- Show the events between a group of classes on an event flow diagram this summarizes events between classes

5.7.4 Building a state Diagram

Prepare a state diagram for each object class with nontrivial dynamic behavior showing the events the object receives and sends. Every scenario or event trace corresponds to a path through the state diagram. Each branch in control flow is represented by a state with more than one exit transition

Start with the event trace diagram that affects the class being modeled trace showing a typical interaction

If the scenario can be repeated indefinitely, close the path in the state diagram

Find loops within the diagram a sequence of events can be repeated indefinitely, these form a loop. Replace finite sequences of events with loops when possible

Merge other scenarios into the state diagram. Attach the new event sequence to the existing state as an alternative path

The hardest thing is deciding at which state an alternate path rejoins the existing diagram. Two paths join at a state if the object "forgets" which one was taken

Beware of two paths that appear identical but which can be distinguished under some circumstances

- After normal events have been considered, add boundary cases and special cases, consider events that occur at awkward times
- Error handling often complicates on otherwise clean and compact program structure, but it must be done
- A state diagram is finished when the object diagram covers all scenarios and the state diagram handles all events
- If there are complex interactions with independent inputs. Organize the dynamic model using a nested state diagram
- Repeat the above process of building state diagrams for each class of objects