UNIT 1 SYSTEM DESIGN

Structure **Page Nos.**

1.0 INTRODUCTION

Object oriented design (OOD) is concerned with developing an object oriented model of a software system to implement the identified requirements. Many OOD methods have been described since the late 1980s. The most popular OOD methods include Booch, Buhr, Wasserman, and the HOOD method developed by the European Space Agency which can yield the following benefits: maintainability through simplified mapping to the problem domain, which provides for less analysis effort, less complexity in system design, and easier verification by the user. OOD also provides reusability, which saves time and costs, and productivity gains through direct mapping to features of Object-Oriented Programming Languages. Object Oriented Development (OOD) has been touted as the next great advance in software engineering. It promises to reduce development time, reduce the time and resources required to maintain existing applications, increase code reuse, and to provides a competitive advantage to organizations that use it. While the potential benefits and advantages of OOD are real, excessive hype has lead to unrealistic expectations among executives and managers. Even software developers often miss the subtle, but profound, differences between OOD and classic software development.

In this Unit we will discuss the basics of object oriented design. We will learn how systems are broken into subsystems, concurrency identification, and how data storage is managed.

1.1 OBJECTIVES

After going through this unit, you should be able to:

- give an overview of object oriented design;
- breaking down system to subsystems;
- explain how a software design may be represented as a set of interacting objects that manage their states and operation;
- identify concurrent object, and explain how to handle concurrency;
- explain methods of storing data and comparison with conventional system design;
- control events between objects; and
- explain boundary condition.

1.2 SYSTEM DESIGN: AN OBJECT ORIENTED APPROACH

Any software systems always tends to change and evolve as technology and business rules change. The evolution of information systems is unavoidable, and it is a natural phenomenon. Organisations need to support systems evolution to take advantage of the new technology and to address the changing business rules. The evolutionary nature of software products requires us to maintain products with their continually changing nature. Not all software products developed are amenable to these fast changes. In this **ever-increasing** competitive business environment, there is hardly any option but *to* ***adopt technology*** that is adaptable to changes.

Software systems designed with **structured design methodology** do not support some of the desired quality attributes such as **reusability, portability** and mapping to the problem domain. Many large organisations find that systems designed with structured approaches are less reusable and less maintainable than those designed with object-oriented approaches.

OOD techniques are useful for development of **large, complex systems**. It has been observed that large projects developed using OOD techniques resulted in a 30% **reduction** in **development** times and a **20% reduction** in **development staff effort,** as compared to similarly sized projects using traditional software development techniques.

Although object oriented methods have roots that are strongly anchored back in the 60s, structured and functional methods were the first to be used. This is not very uncommon, since functional methods are **inspired directly by computer architecture** (a proven domain, well known to computer scientists). The separation of data and program as exists physically in the hardware was translated into **functional methods**. This is the reason why computer scientists got into the habit of thinking in terms of **system functions**. Later it was felt that hardware should act as the **servant** of the software that is executed on it rather than imposing architectural constraints on system development process.

Moving from a functional approach to an object oriented one requires a translation of the functional model elements into object model elements, which is far from being straightforward or natural. Indeed, there is no direct relationship between the two sets, and it is, therefore, necessary to break the model elements from one approach in order to create model element fragments that can be used by the other. In the initial phases of OO design a mixed approach was followed computer scientist tend to use functional approach in analysis phase and OO approach in design phase. The combination of a functional approach for analysis and an object-oriented approach for design and implementation does not need to exist today, as modern object-oriented methods cover the full software lifecycle.

Traditional System Analysis and Design: Traditional System Analysis and Design (SAD) has three fundamental life cycle models. A typical software lifecycle consists of following phases:

- Planning
- Development
- Maintenance

Key Criteria

- a well-defined methodology
- traceability among steps
- documentation
- control

Software Life Cycle

The software design consists of a number of stages, or phases

1. Requirement Phase– determine use need
2. Specification Phase– define requirement
3. Design Phase: design the system
4. Implementation Phase– fabricate the system
5. Testing and Integration Phase– testing the fabricated system
6. Maintenance phase
7. Retirement.

Shortcomings in the Structured approach

- Scalability: The product designed was not scalable
- Maintenance cost: high maintenance cost
- Data and program are separate and difficult to map a real life situation
- Structured methodology treat data and their behaviors (function) separately, this makes it more difficult to **isolate changes**. If certain changes requires then changes in both data structure and algorithms and to be done.

The Proliferation of Object-Oriented Methods

The first few years of the 1990s saw the blossoming of around **fifty different object oriented methods**. This proliferation is a sign of the great vitality of object oriented technology, but it is also the fruit of a multitude of interpretations of exactly what an object is. The drawback of this abundance of methodologies is that it **encourages confusion**, leading users to adopt a 'wait and see' attitude that limits the progress made by the methods. The best way of testing something is still to deploy it; methods are not cast in stone – they evolve in response to comments from their users.
Fortunately, a close look at the dominant methods allows the extraction of a consensus around common ideas. The main characteristics of objects, shared by numerous methods, are articulated around the concepts of class, association (described by *James Rumbaugh*), partition into subsystems (*Grady Booch*), and around the expression of requirements based on studying the interaction between users and systems (***Ivar Jacobson's use cases***).

Finally, well-deployed methods, such as Booch and OMT (Object Modeling Technique), were reinforced by experience, and adopted the methodology elements that were most appreciated by the users.

From SAD to OOAD (Structured Analysis and Design (SAD) to Object Oriented Analysis and Design (OOAD).

We will see here how we can map different models in SAD to different models in OOAD. We will a consider various levels of abstraction through which this is done. The data flow diagram (DFD) in SAD is mapped to Use Case diagram in OOAD. DFD represents a broader model of a system from the process point of view. Hence, DFD cannot be transformed as it is to equivalent representation in UML.

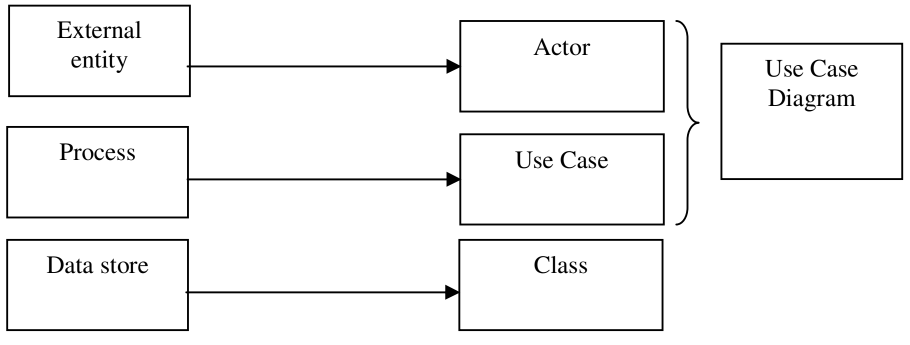


Figure 1: Transformation from SAD to OOAD

The processes are transformed those into Use Case, and the external entity in DFD has similar characteristics to of an actor in OOAD. The data store is transformed into class and part of the data store to attributes of class.

An object-oriented design process

1. Define the context and modes of use of the system
2. Designs the system architecture
3. Identifies the principal system objects
4. Identifies concurrency in the problem
5. Handling boundary conditions
6. Develops design models
7. Specifies object interfaces

Object oriented design is concerned with developing OO model of software systems to implement the identified requirement during the analysis phase.

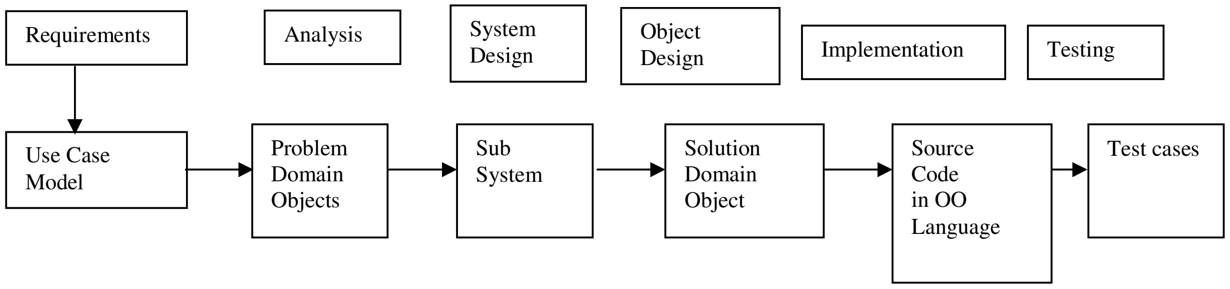


Figure 2: Software life cycle activity

Software developers, data base administrators (DBAs), need to be familiar with the basic concepts of object-orientation. The object-oriented (OO) paradigm is a development strategy based on the concept that **systems should be built from a collection of reusable components called objects**. Instead of separating data and functionality as is done in the structured paradigm, objects encompass both. While the object oriented paradigm sounds similar to the structured paradigm, as you will see in this course material it is actually quite different. A common mistake that many experienced developers make is to applying similar software-engineering principles to OO design. To succeed one must recognize that the OO approach is different than the structured approach.

☞ **Check Your Progress 1**

1) Why were functional methods popular in early days?

…………………………………………………………………………………

…………………………………………………………………………………..

…………………………………………………………………………………

…………………………………………………………………………………

…………………………………………………………………………………

2) What are the shortcomings in structured approach?

…………………………………………………………………………………

…………………………………………………………………………………...

…………………………………………………………………………………

…………………………………………………………………………………...

…………………………………………………………………………………

3) Describe different steps of an object-oriented design process.

……………………………………………………………………………………

..……………………………………………………………………………….

……………………………………………………………………………………

……………………………………………………………………………………

……………………………………………………………………………………

Breaking the system into subsystem involves breaking the problem in to logically independent and interacting subsystems. Now we will discuss how a system is broken into subsystems.

1.3 BREAKING INTO SUBSYSTEMS

Decomposition is an important technique for coping with complexity based on the idea of **divide and conquer**. In dividing a problem into a subproblem the problem becomes less complex and easier to overlook and to deal with. Repeatedly dividing a problem will eventually lead to subproblems that are small enough so that they can be conquered. After all the subproblems have been conquered and solutions to them have been found, the solutions need to be composed in order to obtain the solution of the whole problem. The history of computing has seen two forms of decomposition, **process-oriented** and **object-oriented decomposition.**

Process-oriented decompositions divide a complex process, function or task into simpler sub processes until they are simple enough to be dealt with. The solutions of these subfunctions, then, need to be executed in certain sequential or parallel orders, in order to obtain a solution to the complex process.

Object-oriented decomposition aims at identifying individual autonomous objects that encapsulate both a state and a certain behavior. Each major components of the system is called a subsystem. Then communication among these objects leads to the desired solutions. Decomposition of system in function-data model (SAD) and object oriented decomposition is given below. Although both solutions help deal with complexity, we have reasons to believe that an object-oriented decomposition is favourable because, the object-oriented approach provides for a **semantically richer framework** that leads to decompositions that are more closely related to **entities from the real** world. Moreover, the identification of **abstractions** supports heaving (more abstract) solutions to be reused, and the object-oriented approach supports the evolution of systems better, as those concepts that are more likely to change can be hidden within the objects.

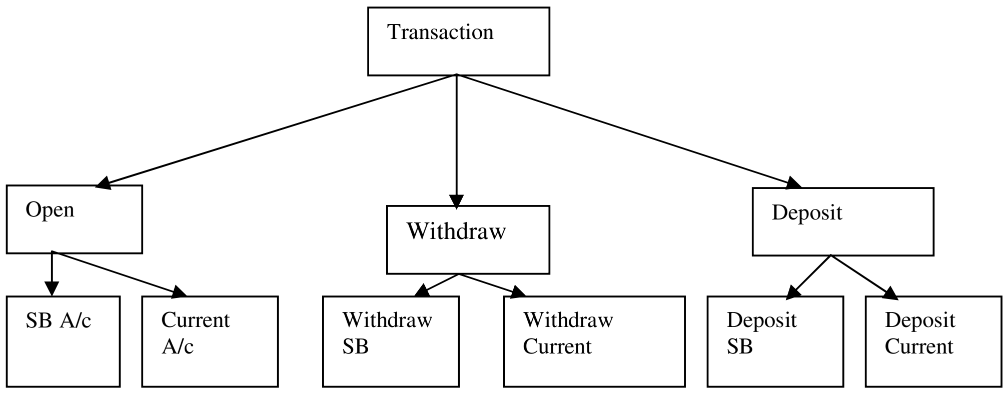


Figure 3: A Function data decomposition (SAD)

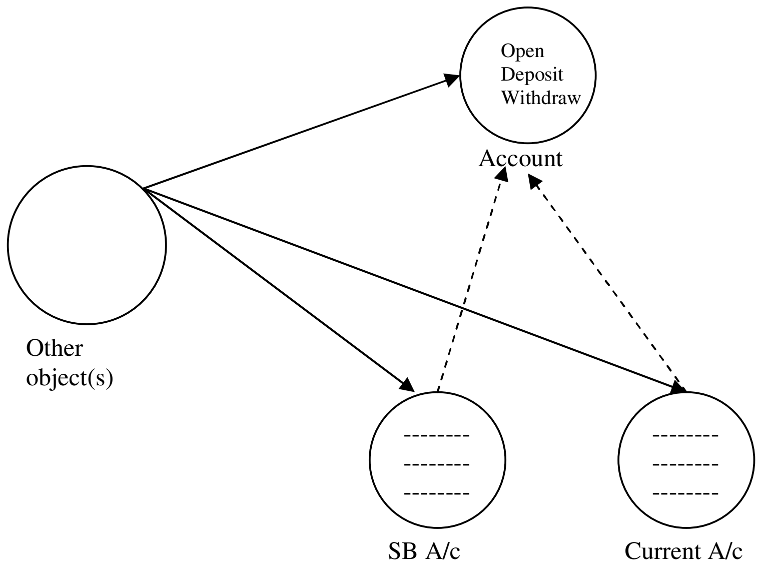
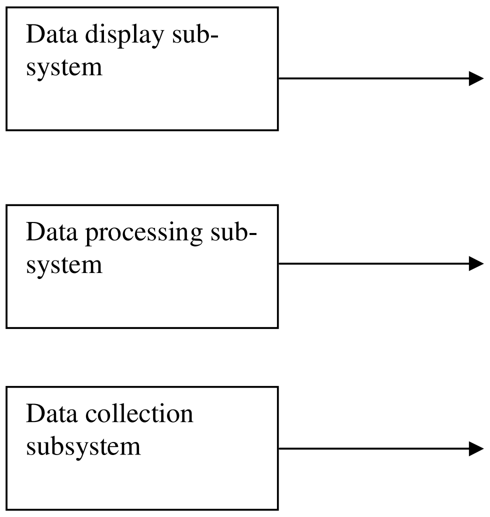


Figure 4: OO Decomposition (OOD)

Object-oriented decompositions of systems tend to be better able to cope with change. Each subsystem has a well-defined interface that communicate with rest of the system. Each of these interfaces defines all form of interaction that are required for proper functioning of the whole system, but the internal implementations are left to the sub-system itself. This is because they manage to encapsulate those items that are likely to change (such as functionality, sequence of behaviour and attributes) within an object and hide them from the outside world. The advantage is that the outside cannot see them, and therefore cannot be dependent on them and does not need to be changed if these items change. Also, object-oriented decompositions are closer to the problem domain, as they directly represent the real-world entities in their structure and behavior. The abstraction primitives built into reuse have a huge potential of
reuse as commonalities between similar objects can be factored out, and then, the solutions can be reused. Finally, object-orientation has the advantage of continuity throughout analysis, design implementation, and persistent representation.

- Object-oriented analysis, design and programming are related but distinct.
- OOA is concerned with developing an object model of the application domain.
- OOD is concerned with developing an object-oriented system model to implement requirements.
- OOP is concerned with realizing an OOD using an OO programming language such as Java or C++.
- Objects are abstractions of real-world/system entities.
- Objects manage themselves.
- Objects are independent and encapsulate state and representation.
- System functionality is expressed in terms of object services.
- Shared data areas are eliminated.
- Objects communicate by message passing.
- Objects may be distributed and may execute sequentially, or in parallel.

The system can be decomposed into two-layer architecture (referred as layer) vertical and decomposition (referred as partition).



Data display layer where the objects of the sub-system are concerned only with presentation of the user interface

Data processing layer where objects check the integrity of data that goes to the database.

The other types of decomposition called partition are commonly found in operating system. Most of the large system may require a mixture of both types of decomposition.

Now, let us see the basic advantages of decomposition.

The Advantages of Decomposition

1. Separate people can work on each subsystem.
2. An individual software engineer can specialize in a domain.
3. Each individual component is smaller, and therefore easier to understand and manage.
4. Part of the subsystem can be replaced or changed without having to replace or extensively change other subsystems.

Concurrency identification is very challenging in nature. In the next section, we will discuss objects concurrency identification.

1.4 CONCURRENCY IDENTIFICATION

While designing the analysis, model we map the real world model into our analytical model. Real life objects are concurrent in nature, but all **design model objects are not concurrent in nature** as a **single process may support multiple objects.**

Let us see what concurrency actually is: **Concurrency in objects can be identified by the way they change state**. Current objects can change state **independently**. Aggregation implies concurrency. Concurrency in OOAD study is described and studied in **dynamic modeling**.

One of the important issues in system design is to find the concurrency in objects. Once we identify non-concurrent (mutually exclusive) objects, we can fold all the objects together in **one thread of control, or process**. On the other hand, if the objects are concurrent in nature we have to assign them to, **different thread of control**. For example, withdraw and deposit operations related to a bank account may be executed in parallel (concurrently).

- A thread of control is a path through a set of state diagrams on which a single object is active at a time.
- Objects are shared among threads, that is, several methods of the same object can be active at the same time.
- Thread splitting: Object sends a message but does not wait for the completion of the method.

Identification of concurrency: Concurrency is identified in a dynamic model. Two objects are said to be concurrent (parallel) if they can receive events **at the same**

time. Concurrent objects are required to be assigned to different threads of control. We will see how if is used in this example:
Example:

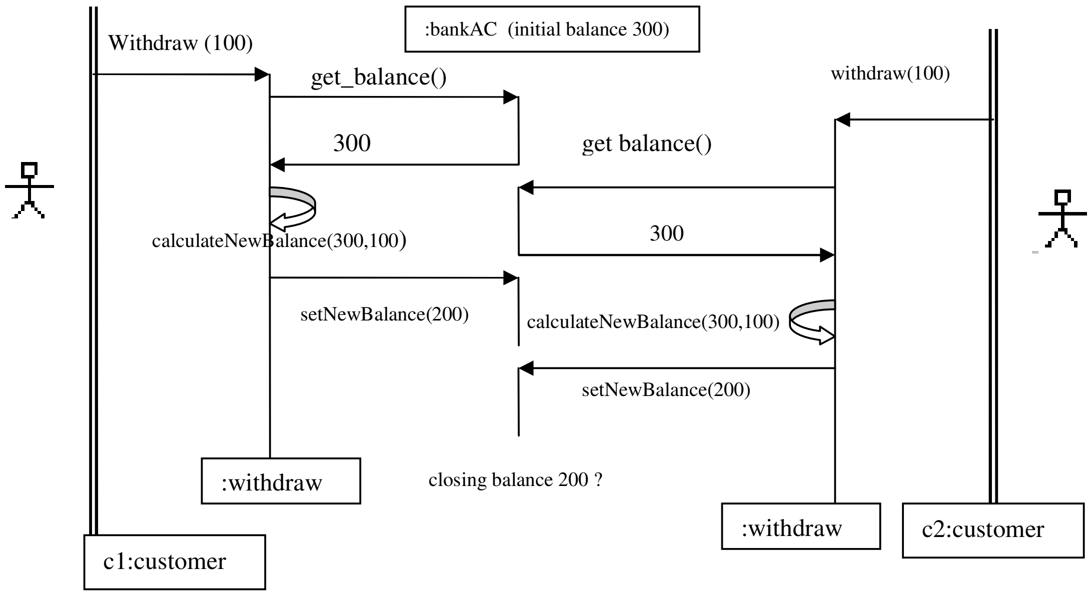


Figure 6: Concurrency without synchronization

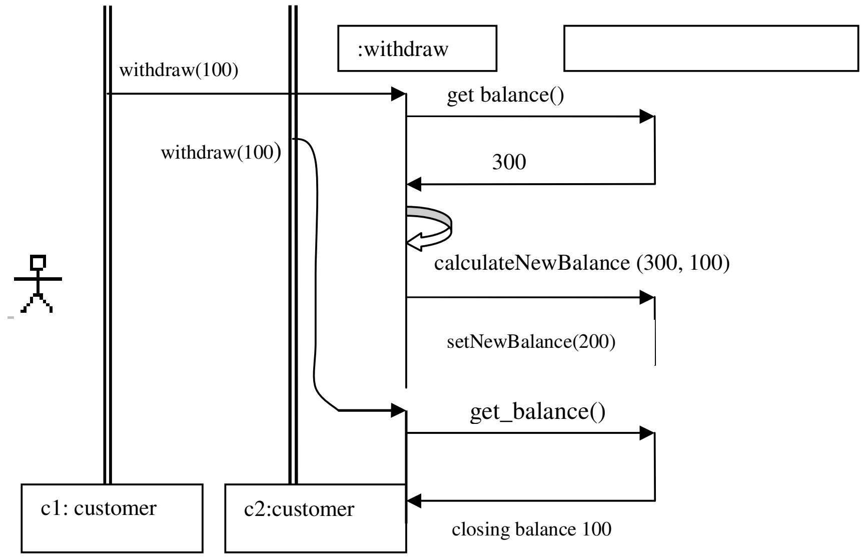


Figure 7: Concurrency with synchronization

Refer to the *Figure 7* if an object must perform two or more activities concurrently, then the internal steps of the process must be synchronized. Both the synchronized activity must complete before the object performing the concurrent activity can go to the next step.

Concurrency issues

- **Data integrity:** Threads accessing the same object need to be synchronized, for example: banking account.
- Deadlock: One or more threads in the system are permanently blocked
  Example: Thread A waiting on Thread B, which is waiting on Thread A.
- Starvation: A thread is not getting enough resources to accomplish its work

  Example: All requests from one user are being handled before another users requests.

How to handle concurrency*:*

Mechanisms
- Locks
- Semaphores
- Monitors
- Synchronized methods

Methods
- Deadlock avoidance
- Verification
- Simulation

Key:

1) Develop a clear strategy for dealing with all concurrency issues during system design.
2) Concurrency must be dealt with during the design process as dealing with concurrency after the system is implemented is difficult.

A detailed discussion on this topic will be there in MCS 041.

☞ **Check Your Progress 2**

1) What are the advantages of decomposing a system?

   ……………………………………………………………………………………

   ……………………………………………………………………………………

   ……………………………………………………………………………………

2) Differentiate between object oriented decomposition and structured decomposition?

   ……………………………………………………………………………………

   ……………………………………………………………………………………

   ……………………………………………………………………………………

3) How is concurrency identified?

   ……………………………………………………………………………………

   ……………………………………………………………………………………

   ……………………………………………………………………………………

   ……………………………………………………………………………………

   ……………………………………………………………………………………

Data storage is a very important stored aspect involve in any systems. In the next section we will discuss management of data.

1.5 MANAGEMENT OF DATA STORE

Every system irrespective of their nature of application needs to store permanent data for subsequent use in problem solving. Some objects in the models need to be **persistent**, to store the state of the object permanently in database. Most systems need **persistent data** which is not lost when the system closes down. These data are stored in file system, in a or database. Object-oriented applications often use relational databases to provide persistence.

Designer needs to:

1. Identify what data needs to be persistent
2. Design a suitable database schema for the database.

Persistent classes are shown using tagged value in UML diagram.

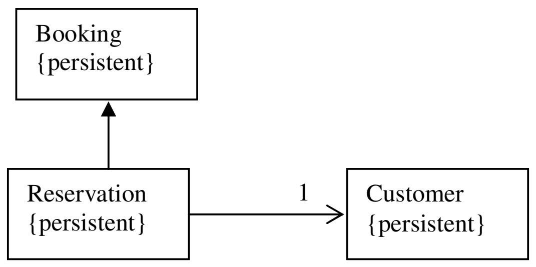


Figure 8: A UML diagram showing persistent class

For example, in reference to *Figure 8*, we must save all information related to customers and booking details.

In most of the cases, persistent class maps to one relational table (leaving aside the inheritance issue, for the moment). In the simplest model, a class from the logical model maps to a relational table, either in whole, or in part. The logical extension of this is that a single object (or instance of a class) maps to a single table row. Persistent object can be stored with one of the following:

Files

- Cheap, simple, permanent storage
- Low level (Read, Write)
- Applications must add code to provide a suitable level of abstraction.

Relational database

- Standardized, easy to port
- Supports multiple writers and readers
- Mature technology

Object database

- One-to-one mapping from the analysis model
- Associations are directly represented
- Slower than relational databases for complex queries

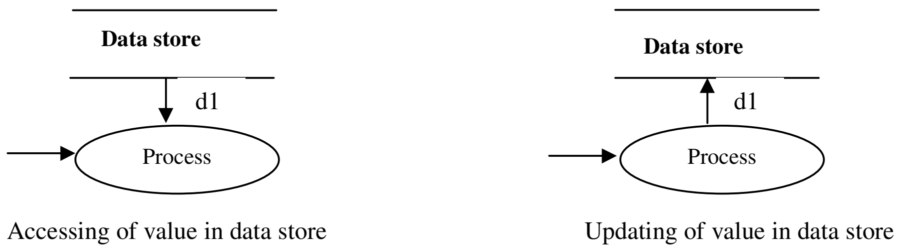

Data store
d1
Process
Accessing of value in data store
Data store
d1
Process
Updating of value in data store

Figure 9: Functional Model notation of data store

The advantage of using a Database Management System for a data store is that databases have mechanisms for **describing data, managing persistent storage and for providing a backup and recovery mechanism**. It also provides concurrent access to the stored data with an appropriate locking mechanism. Most of the **DBMS** contains information about data in the form of a data dictionary. Most commercial RDBMS come with an "**Object-Relational**" extension which implements an object database on top of a RDBMS.

Issues when Selecting a Database

- **Storage space requirement**: A database requires about triple the storage space of actual data.
- **Response time**: The response time of the database for I/O or communication bound (in case of distributed databases) request.
- **Locking modes pessimistic locking**: Lock before accessing an object and release when object access is complete.

- **Optimistic locking:** Read and writes may freely occur (high degree of concurrency).
- **Administration**: Modern DBMS requires specially trained support staff to set up security policies, manage the disk space, prepare backups, fine-tune the performance, and monitor performance.
- How often is the database accessed?
- What is the expected request (query) rate? In the worst case?
- What is the size of the typical request and of the worst case requests?
- Need for data to be archived.

Table 1

Relational Databases	Object-Oriented Databases
• Based on mathematical principles called relational algebra • Data are represented by a two dimensional table with columns and rows • Implements standard query language called SQL • Most RDBMS supports various constraints, like referential integrity.	• Supports all fundamental object modeling concepts: Classes, Attributes, Methods, Associations, Inheritance • Support for complex objects • Provides for mapping an object model to an OO-database • Determine which objects are persistent • Perform normal requirement analysis and object design • Create single attribute indices to reduce performance bottlenecks.

Most of the object oriented system use relational database to store persistent data. The advantages and disadvantages of OO database are compared in Table 2.

Table 2

Advantages of OO Database	Disadvantages OO Database
• Supports all fundamental object modeling concepts like Classes, Attributes, Methods, Associations, Inheritance • Maps an object model to an OO-databases • Determine which objects are persistents • Performs normal requirement analysis and object design • Creates single attribute indices to reduce performance bottlenecks • Supports complex objects. • Extensibility of data types. • Improves performance with efficient caching. • Versioning • Faster development and easy maintenance through inheritance and reusability.	• Strong opposition from the established players of relational database • Lacks rigorous theoretical foundation. • Retrogressive to the old pointer systems • Lacks standard ad hoc query language like SQL. • Lack of standards affects OO database design • Lack of business data design and management tools • Steep learning curve.

Other System Design Issue

- How to realise the subsystems: through hardware or software?
- How is the object model mapped on the chosen hardware software?
- Mapping objects onto hardware: processor, memory, I/O.
- Mapping associations onto networks: Connectivity.

Much of the difficulty of designing a system comes from fulfilling the restriction imposed by hardware and software constraints. This may include cases where certain throughput has to be guaranteed for a system, and where certain response time has to be guaranteed.

1.6 CONTROLLING EVENTS BETWEEN OBJECTS

Event is the specification of a significant occurrence that has a location in time and **space**.

Examples of events are mouse click and flight leaving from an airport. An event does not have a **fixed duration.** Each thing that happens modeled as an event. After an event, objects change their state, and these are represented by a state diagram.

Events are classified as four types in UML

1. Signals
2. Calls
3. Passing of Time
4. Change in State

Events also include inputs, decisions, interrupts and actions performed by users or any external device. Every event has a sender and receiver. In most of the cases the sender and receives are the same object. A state without a predecessor and successor are ambiguous, and care should be taken to represent initiations and termination of events. Events that have same effect on the control flow must be grouped together even if their value differs. The events are to be allocated to the object classes that send/receive it.

Most of the design issues of systems are concerned with steady-state behavior. However, the system design phase must also address the initiation and finalization of the system. This is addressed by a set of **new uses cases** called administration use cases. Now, let us discuss how these issues are handled.

1.7 HANDLING BOUNDARY CONDITION

These are some conditions which to be handled in any system initialization and termination

- Describes how the system is brought from a non-initialized state to steady-state (“startup use cases”). It describes normal operations like start-up, shutdown, reconfiguration, restart/reset, backup, etc.

- Describes what resources are cleaned up, and which systems are notified upon termination (“termination use cases”).

Configuration

- Describes how the system is adapted to a local installation.

Failure

- Unplanned termination of the system.
- Many possible causes: failure of system hardware, bugs, operator errors, external problems (power supply).
- Good system design foresees fatal failures ("failure use cases").

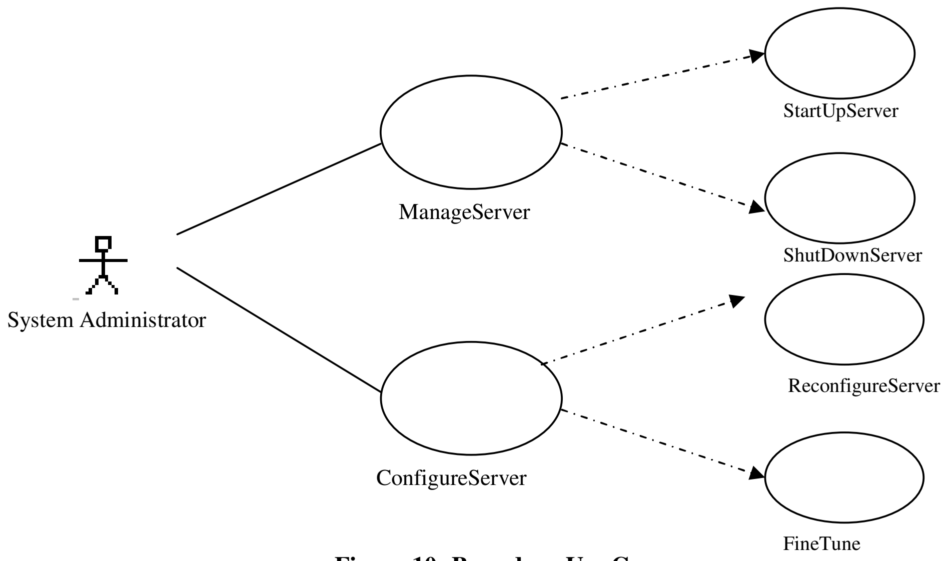


Figure 10: Boundary Use Case

Operation	**Actor**	**Constraints**
Start up	System Administrator / Operator	Availability of power, and no system fault
Shut down	System Administrator / Operator	No user is active, and data has been saved
Fine Tune	System Administrator	No user is active, database offline
Reconfigure	System Administrator	No user is active, system resources are available

While defining the boundaries of a system we must ascertain what system entities one does and does not have control over. The level of control is determined for all identified internal and external entities. The control status can be total, partial, or none.

☞ Check Your Progress 3

1) What is persistency?

……………………………………………………………………………………

……………………………………………………………………………………

……………………………………………………………………………………

2) Why, generally, does an object-oriented system use a relational DBMS?

……………………………………………………………………………………

……………………………………………………………………………………

……………………………………………………………………………………

3) Identify and name a few boundary processes.

……………………………………………………………………………………

……………………………………………………………………………………

……………………………………………………………………………………

4) Draw a use case diagram for a typical Flight Reservation System. Identify use case and actor.

…………………………………………………………………………………

…………………………………………………………………………………

…………………………………………………………………………………

1.8 SUMMARY

OOD techniques are useful for the development of large, complex systems. Moving from a functional approach to an object-oriented one requires a translation of the functional model elements into object model elements, which is far from being straightforward. The high-level system design approach involves breaking down the system into simple and relatively independent sub systems for better manageability. Systems can be proportioned to horizontal or vertical partitions. Concurrency is inherent to objects, and concurrent objects cannot be folded into a single thread of control. Concurrency must be dealt with during the design process as dealing with concurrency after the system is implemented is difficult.

Most systems need **persistent data** which is not lost when the system closes down. Data can be stored in flat files or DBMS. Object-oriented databases provide support for all fundamental object modeling concepts like Classes, Attributes, Methods, Associations, and Inheritance. But unlike RDBMS, object-oriented databases lack theoretical foundation and standards. Although system design is concerned with the steady state behaviors of the system, the system must be designed to handle boundary conditions. Boundary use cases are useful to analyse boundary conditions.

1.9 SOLUTIONS/ ANSWERS

Check Your Progress 1

1) Functional methods are inspired directly by computer architecture, and thus the popular among computer scientist in the early days. The separation of data and program as it exists physically in the hardware was translated into the functional methods.

2) Structured methodology treats data and their behaviors (functions) separately, and this makes it more difficult to isolate changes. If changes are required in the software then one has to change both the data structure and the algorithms and these changes subsequently require changes in the algorithms where this data structure is used as well.

3) The broad steps of an object-oriented design process are:

   a. Define the context and modes of use of the system

   b. Design the system architecture

   c. Identify the principal system objects

   d. Identification of concurrency in the problem

   e. Handling boundary condition

   f. Develop design models

   g. Specify object interfaces.

Check Your Progress 2

1) The advantages of decomposing a system into subsystems are that after decomposition, each individual components become smaller, and easy to manage. Changes in these subsystem can be effected without extensively

making changes in other subsystem. Decomposition it also allows software engineer to specialize in a particular domain of the system.

2) Process-oriented (structural) decomposition divides a complex process, function or task into simpler subprocesses until they are simple enough to be dealt with. The solutions of these subfunctions then need to be integrated and executed in certain sequential or parallel orders in order to obtain a solution to the large complex system. On the other hand object-oriented decomposition aims at identifying individual autonomous objects that encapsulate both a state and certain behavior. Each major component of the system is called subsystem. Then, communication among these objects leads to the desired solutions. Object-oriented decompositions of systems tend to be better able to cope with change. Each subsystem has well-defined interfaces that communicate with rest of the system

3) Concurrency in objects can be identified by the way **they change state**. Current objects can change state independently. Aggregation implies concurrency in the system.

Check Your Progress 3

1) Persistency ensures that data is stored, and that after the object is no longer available (program stops running), the data will be, available to other users, as and when needed.

2) The reasons for using RDBMS for OO systems could be any combination of the following:

   a. Many organizations have existing relational databases containing existing business data.

   b. Most commercial RDBMS come with an "Object-Relational" extension which implements an object database on top of a RDBMS.

   c. Purely object databases are too complicated to use, and lack standards like SQL.

3) System startup, shutdown, system failure.

4) Actors: Passenger, Ticket clerk.

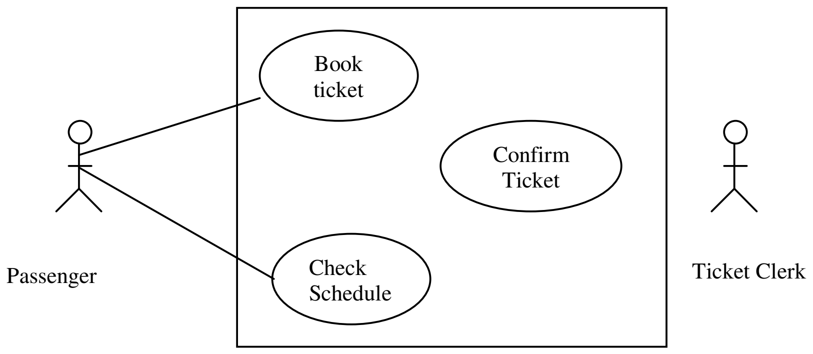


Figure 11: Use case for Book Ticket, check schedule and confirm Ticket