Transaction Manager: A system concerned with storing and updating data, often including concurrent access from different physical locations.

7.4 Object Design

The object design phase determines the full definitions of the classes and associations used in the implementation, as well as the interfaces and algorithms of the methods used to implement operations. The object design phase adds internal objects for implementation and optimize data structure and algorithms. Object design is analogues to the preliminary design phase of the traditional software development life cycle.

During the object design phase, the designer must perform the following steps:

(*i*) Combine the three models to obtain operations on classes.

(*ii*) Design algorithm to implement operations.

(*iii*) Optimize access paths to data.

(*iv*) Implement control for external interactions.

(*v*) Adjust class structure to increase inheritance.

(*vi*) Design associations.

(*vii*) Determine object representation.

(*viii*) Package classes and associations into modules.

7.4.1 Grouping the Three Models

After completion of analysis phase, we have the three models: Object model, dynamic model and functional model, but the object model is the main framework around which the design is constructed.

The designer must convert the actions and activities of the dynamic model. In making this conversion, we start the process of mapping the logical structure of the analysis model into a physical organization of a program.

The designer must transfer operations from the functional and dynamic model onto the object model for implementation. A process from the functional model becomes an operation on an object. An event from the dynamic model may also become an operation on an object, depending on the implementation of control.

7.4.2 Designing Algorithms

Each operation from the analysis model must be assigned an algorithm that implements it clearly and efficiently, according to the optimization goals selected during system design. The analysis specification tells *what* the operation does from the viewpoint of its client, but the algorithm shows *how* this operation is does.

An algorithm may be subdivided into calls on simpler operations and so on recursively, until the lowest level operations are simple enough to implement directly without further refinements.

The designer of algorithm must:

- Choose algorithms that minimize the cost of implementing operations.
- Select data structures appropriate to the algorithms.
- Define new internal classes and operation as necessary.
- Assign responsibility for operations to appropriate classes.

7.4.3 Design Optimization

In the design optimization, we optimize the model which is made with the help of system design and analysis. The analysis model contains the logical information about the system, while the design model must add details to support efficient information access. For the efficiency and clarity, we must extend and restructure the model. The original information is not discarded, but new redundant information is added to optimize access paths and preserve intermediate results that would otherwise have to be recomputed. Algorithm can be rearranged to reduce the number of operations to be executed.

During the design optimization, the designer must:

- Add redudant associations to minimize access cost and maximize convenience of accessing.
- Rearrange the computation for greater efficiency and rearrange the order of execution of appropriate algorithm.
- Save derived attributes to avoid recomputation of complicated expressions.

7.4.4 Implementation of Control

The designer must refine the strategy for implementing the state-event models present in the dynamic model. In the system design, we have chosen the basic strategy for realizing the dynamic model and in the object design we must flesh out this strategy.

There are three basic approaches to implementing the dynamic model (as explained in Section 7.3.6).

(*i*) Using the location within the program to hold state (procedure-driven system)
(*ii*) Direct implementation of a state machine mechanism (event-driven system).
(*iii*) Using concurrent system.

7.4.5 Adjustment of Inheritance

During object design, the definitions of internal classes and operations can be adjusted to increase the amount of inheritance. These adjustment include modifying the argument list of a method, moving attributes and operations from a class into a superclass, defining an abstract superclass to cover the shared behavior of several classes, and splitting an operation into an inherited part and a specific part. Delegation should be used rather than inheritance when a class is similar to another class but not truly a subclass.

Thus to increasing the amount of inheritance designer should:

- Rearrange and adjust classes and operations to increase inheritance.
- Abstract common behavior out of groups of classes.
- Use delegation to share behavior when inheritance is symmetrically invalid.

Delegation is also technique to share behavior among the classes.

7.4.6 Design of Associations

Association provides access paths between objects. There are conceptual entities useful for modeling and analysis. To make intelligent decisions about association, we first need analyze the way they are used and the implement each association as a distinct object or by adding object valued attributes to one or both classes in the association.

Associations subsume many implementation techniques under a single uniform notation during analysis, but they can be implemented as pointer with in objects or distinct objects depending on their access patterns. An association traversed in a single direction can be implemented as an attribute pointing to another objects or a set of objects, depending on the multiplicity of the association.

A bidirectional association can be implemented as a pair of pointers, but operations that update the association must always modify both directions of access. Associations can also be implemented as association objects.

7.4.7 Object Representation

For the exact representation, the designer must choose when to use primitive types in representing objects and when to combine groups of related objects.

Classes can be defined in terms of other classes, but everything must be implemented in terms of built-in primitive data types (such as integers, strings and enumerated type) supplied by the programming language. Some classes can be combined.

7.4.8 Physical Packaging

Programs are made of discrete physical units that can be edited, compiled, imported, or otherwise manipulated. Programs must be packaged into physical modules for editors and compilers as well as for the convenience of programming terms. Object-oriented languages have various degrees of packaging.

Packaging involves the following issues:

- Hiding internal information from outside view (using private and protected data).
- Coherence of entities (entities should be organize in a coherent way).
- Constructing physical modules (entities should be organize in a common theme).

7.4.9 Object Design Documentation

Design decisions should be documented by extending the analysis model, by adding detail to the object, dynamic, and functional models. Implementation constructs are appropriate, such as pointers (in the object model), structured pseudocode (in the dynamic model) and functional expressions (in the functional model).

The design document should be an extension of the Requirements Analysis Document. Thus the design document will include a revised and much more detailed description of the object model, in both graphical form (object model diagrams) and textual form (class descriptions).

The functional model will also be extended during the design phase, and it must be kept current.

The dynamic model is implemented using an explicit state control or concurrent tasks then the analysis model or its extension is adequate. If the dynamic model is implemented by location within program code, then structured pseudocode for algorithms is needed.

7.5 Implementation

In this section we will discuss the how can you implement your designed model in software development. The software development of system can be implemented using:

(*i*) Programming languages

(*ii*) Database management system

Writing code is an extension of the design process. Writing code should be straightforward, almost technical, because all the difficult decisions should already have been made during design. The code should be a simple translation of the design decisions into the syntaxes of a particular language.

7.5.1 Using Programming Languages

Implementation of the system using programming language is a traditional way, most of the programming languages are capable of expressing the three aspects of software specification:

- Data structure
- Dynamic flow of control
- Functional transformation

Data structure provides a way of data declaration, is a subset of a language. The statements that are used to declare data structures are commonly non-procedural part of the languages.

Flow of control may be expressed either procedurally (conditionals, loops and calls) or non-procedurally (rules, constraints, tables and state machines). Traditional languages are purely procedural, although the programmer can implement non procedural constructs as data.

Functional transformations are expressed in terms of the primitive operations of the language, as well as calls to subprograms. Most procedural languages are similar in the kinds of functionality they supported do not differ greatly from FORTRAN.

There are two types of programming languages for implementation of an object-oriented design:

(*i*) Object-oriented languages

(*ii*) Non-object oriented languages

(*i*) Object-Oriented Languages

Implementation of an object-oriented design is easiest using an object-oriented language, but even these languages vary in their degree of support for object-oriented concepts. The object-oriented language is the most natural implementation target for an object-oriented design.

Good programming style is important to maximize the benefits of object-oriented design and programming; most benefits came from greatly reduced maintenance and enhancement costs and in reuse of the new code on future projects. Object-oriented programming style guidelines include conventional programming style as well as uniquely applicable to object-oriented concepts such as inheritance. There are number of languages that support object concepts such as inheritance. There are number of languages that support object oriented programming such as C++, small talk and Java.

Advantages of Object-oriented Languages

- **Reusability**

  Once a class or code has been written, created and debugged, it can be distributed to other programmers for use in their own programs. This is called reusability. It is similar to the way a library of functions in a procedural language can be incorporated into different programs. The idea of reusability can be active by inheritance.

- **Extensibility**

  Most software is eventually extended. Extensibility is enhanced by
  - Encapsulation of classes and methods.
  - Minimizing dependencies between classes and methods.
  - Using methods to access attributes of other classes.
  - Distinguishing public from private operations on a class.

  The techniques for reusability enhance extensibility.

- **Robustness**

  A program is said to robust, if it does not fail even if it receives improper parameters. Robustness against internal bug (error) may be trade-off against efficiency. Robustness against user errors should never be sacrificed. Program should always protect against user and system errors.

  It provides the facility of:
  - Protection against errors.
  - Optimization after the program execution.
  - Validation of arguments.
  - Avoidation of predefined limits.
  - Service of debugging.
  - Performance monitoring.

- **Programming-in-the-Large**

  Programming-in-the-large refers to writing large, complex programs with teams of programmers. The writing of large programs with team of programmer requires more discipline, best documentation and communication than one-person or small projects. It provides the use of exactly the same names as in the object model. You should make methods more readable and understandable.

(ii) Non-Object-Oriented Language

Object-oriented concept can be mapped into non-object-oriented language constructs. There is not a big problem (excluding issue of expressiveness) to implement object oriented design, because programming languages are eventually converted to machine language. Use of a non-object-oriented language requires greater care and discipline to preserve the object oriented structure of the program. The most non-object-oriented languages, like C, Ada, Pascal and Fortran provides the implementation of object-oriented design.

Implementing an object-oriented design in a non-object-oriented language requires basically the same steps as implementing a design in an object oriented language.

The programmer using a non-object-oriented language must map. Object-oriented concepts into the target language, whereas the compiler for a object-oriented language performs such a mapping automatically. The steps required to implement a design are:

- Translate classes into data structure (or records).
- Pass arguments to methods (or functions).

- Allocate storage for objects.
- Implement inheritance in data structure.
- Implement method resolution.
- Implement associations.
- Deal with concurrency.
- Encapsulate internal details of classes.

7.5.2 Using Database System

A database management system (DBMS) is a computer program that is designed to provide genera purpose functionality for storing, retrieving and controlling access to permanent data. A DBMS protects data against accidental loss and makes it available for sharing. It provides the managemen of a permanent, self-descriptive repository of data, called a *database* and is stored in one or more files.

The Advantage of DBMS:

- **Crash Recovery:** The database is protected from hardware crashes, disk media failures and some user errors.
- **Security:** Data can be protected against unauthorized users to read and write access.
- **Integrity:** You can specify the rules and conditions that data must satisfy. A DBMS can control the quality of its data over and above facilities that may be provided by application programs.
- **Sharing between Users:** The same database can be access by multiple users at the same time.
- **Sharing between Applications:** Multiple application programs can read and write data from/ to the same database. A database is natural medium that facilitates communication between free-steady programs.
- **Extensibility:** Without interrupting the existing program data can be add to the database. Data can be reorganized for faster performance.
- **Data Distribution:** The database may be partitioned across various users, sites, organization and hardware platforms.

Several DBMS paradigms are available:

- Hierarchical DBMS
- Network DBMS
- Relational DBMS
- Object-oriented DBMS

Hierarchical and Network DBMS brings the conceptual DBMS close to the underlying physical data structures.

Relational DBMS present the database at a higher level of abstraction than hierarchies and networks are easier to use. Relational DBMS implementations are improving in performance as the mature and use smaller optimization techniques.

The object-oriented DBMS provide full supports for all the three models of the object design. The object classes used by the programming language are the classes used by the DBMS, because their models are consistent, there is no need to transform the program's object model to something unique for the database manager.

Object-oriented database have the ability to model all the models directly with in the database supporting a complete problem/solution modeling capability. It also provides the features of inheritance, polymorphism and dynamic binding.

7.6 Comparison of Methodologies

There are several popular software engineering approaches for developing the softwares. And most of the approaches are based on the data flow diagrams. In this section we are presenting two other approaches to compare with OMT methodology, which are:

- Structured Analysis/Structured Design (SA/SD)
- Jackson Structured Design (JSD)

Basically our aim is to clearly identify the major differences and similarities between the OMT and other methods logics, because each approach has their own strengths and weaknesses.

7.6.1 Comparison with SA/SD

SA/SD methodology contains much common features as OMT. Both methodologies use similar modeling constructs and support the three orthogonal views of a system. SA/SD includes a variety of notations for formally specifying software. During the analysis phase, data flow diagrams, process specifications, a data dictionary, state transition diagrams, and entity relationship diagram are used to logically describe a system. In the design phase, details are added to the analysis models and the data flow diagrams are converted into structure chart descriptions of programming language code.

The difference between SA/SD and OMT is primarily a matter of style and emphasis.

- In SA/SD approach, the functional model dominates, the dynamics model is next important, and the object model least important.
- SA/SD organizes a system around procedures, while OMT organizes a system around real-world objects, or conceptional objects that exist in the user's view of the world.
- SA/SD is useful for problems where functions are more important and complex than data.
- An SA/SD design has a clearly-designed system boundary, across which the software procedures must communicate with the real world, so it can be difficult to extend a SA/SD design to a new boundary. While it is much easier to extend an object oriented design; one merely adds objects and relationship near the boundary to represent objects that existed previously only in the outside world.
- An object oriented design is more extensible, provides better traceability, and better integrates database and programming code.

7.6.2 Comparison with JSD

Jackson Structured Development (JSD) is another methodology of software engineering. JSD has a different development style than SA/SD or OMT. JSD does not distinguish between analysis and design and instead emphasizes on both phases together as specification. JSD divides system development into two stages: specification, then implementation. A JSD model describes the real-world in terms of entities, actions and ordering of actions.

JSD software development consists of six sequential steps:

(*i*) Entity action
(*ii*) Entity structure
(*iii*) Initial model
(*iv*) Function
(*v*) System timing
(*vi*) Implementation

— *JSD approach uses graphical models, but JSD is less graphically oriented than SA/SD and OMT.*

— *JSD approach is complex and difficult to fully comprehend. JSD is complex because it was specifically designed to handle difficult real time problems. For these problems, JSD may produce a superior design and be worth the effort.*

— *JSD places more emphasis on action and less on attributes than OMT.*