**Module-3**

**Chapter-8**

**Analysis**

**8.1 Overview of analysis**

- Analysis is the first step of Object Modeling Technique(OMT) methodology

- Analyst tries to understand about the problem

- Analysis determines what is to be done and not how it is implemented

- Analysis begins with a problem statement

- Analysis is not a mechanical process

- Most problem statement lack essential information which must be obtained from the requestor

- The analyst must communicate with the requestor to clarify ambiguities

- The analysis model facilitate precise communication

**8.2 Object Modeling**

- The first step in analyzing the requirement is to construct an object model

- The object model describes real-world object classes and their relationship to each other

- Information for object model comes from problem statement, expert knowledge of application domain and general knowledge of real world

- Identify classes and associations first and add attibutes

- Then combine and organize classes using inheritance

- Add operations to classes

- Steps in constructing an object model are:

* Identify objects and classes
* Prepare a data dictionary
* Identify associations
* Identify attributes of objects and links
* Organize and simplify object classes using inheritance
* Verify that access paths exist for likely queries
* Iterate and refine the model
* Group classes into modules

**8.2.1 Identifying object classes**

- First step in constructing an object model is to identify relevant object classes from

application domain

- Object include physical entities such as house, employees and machines

- Eg: If you are building a cataloging and check out system for library, identify different kinds of material such as books, magazines, newspaper etc. Then organize them into broad categories by looking similarities and differences

**8.2.2 Keeping the right classes**

- **Redundant classes**:- If two classes express the same information the most descriptive name should be kept

- **Irrelevant classes**:- If a class has little or nothing to do with the problem, it should be eliminated

- **Vague classes**:- Class should be specific. Some tentative classes may be too broad in scope. Eg: Record keeping provision is vague. In ATM problem, this is part of Transaction

- **Attributes**:- Names that describe individual objects. Eg: Name, age, weight

- **Operations**:- Transformations that are applied on or by objects in a class. Eg: Open, close, are operations of class window

- **Roles**:- The name of a class should reflect its intrinsic nature

- **Implementation constructs**:- Constructs should be eliminated from analysis model. They may be needed later during design

**8.2.3 Preparing data dictionary**

- Write a paragraph describing each object class

- Describe the scope of class within the current problem

- Also describe attributes, operations and association

**8.2.4 Identifying association**

- Identify association between classes

- Reference from one class to another is an association

- Eg: Relate class Person and class Company by association Works-for

- Association shows dependencies between classes at same level of abstraction

- Association include physical location, communication or ownership

**8.2.5 Keeping the right association**

- Discard unnecessary and incorrect association using following criteria

* **Association between eliminated classes**:- If one of the classes in the association has been eliminated the association must be eliminated
* **Irrelevant or implementation association**:- Eliminate any association that are outside the problem domain or deal with implementation constructs
* **Actions:**- An association should describe the structural property of application domain. Eg: ATM accepts cash card describes part of interaction cycle between an ATM and a customer
* **Ternary association**:- Most association between three or more classes can be decomposed into binary association or phrased as qualified association
* **Derived association**:- Omit associations that are defined in terms of other association using dotted lines
* **Misnamed association**:- Names are important for understanding and should be chosen with great care
* **Role names**:- Add rolenames where appropriate. The rolename describe the role that a class in association plays from the point of view of other class
* **Qualified association**:- Qualifier distinguishes objects on many side of association. It reduces multiplicity
* **Multiplicity:-** Describes one or many
* **Missing association**:- Add any missing association that are discovered

**8.2.6 Identifying attributes**

- Identify object attributes

- Attributes are properties of individual objects such as name, weight, velocity or color

- Attributes should not be objects

- Consider attributes that directly relate to a particular application

- Important attributes should be collected first

- Fine details can be added later

- During analysis avoid attributes for implementation

- Derived attributes should be omitted or clearly labeled

- Eg: age is derived from birth date and current time

- Derived attributes should not be expressed as operations Eg: get-age

- Link attributes should be identified

- Derived attributes can be useful in abstracting meaningful properties of an application, but they should be clearly distinguished from base attributes

**8.2.7 Keeping the right attributes**

- Eliminate unnecessary and incorrect attributes with the following criteria:

* **Objects**:- If independent existence of an entity is important, rather than just its value then it is an object. Eg: Boss is an object and salary is an attribute
* **Qualifiers**:- If the value of an attribute depends on particular context then restate attribute as qualifier
* **Names**:- Names are modeled as qualifiers. Name is an object attribute when it does not depend on context
* **Identifiers:**- Used for referencing an object
* **Link attributes**:- If a property depends on link then the property is an attribute of link
* **Internal values**:- If an attribute describes the internal state of an object that is invisible outside the object then eliminate it from analysis
* **Fine detail**:- Omit minor attributes which affect operations
* **Discordant attributes**:- An attribute that seems completely different from and unrelated to all other attributes may indicate a class that should be spilt into two distinct classes

**8.2.8 Refining with inheritance**

- Next step is 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 super class
* By refining existing classes into specialized subclasses
* For each generalization define a superclass to share common features
* Inheritance can be discovered by searching for classes with similar attributes, associations or operations
* Avoid excessive refinement
* Multiple inheritance may be used to increase sharing when necessary
* Secondary superclasses add orthogonal details
* When the same association name appears more than once with same meaning try to generalize the associated class

**8.2.9 Testing access paths**

- Trace access path through the object model diagram to see if they yield sensible result

- Eg: ATM: The card cannot identify an account. So the user must choose an account whether its saving or current

**8.2.10 Iterating object modeling**

- Object model is rarely correct after a single pass

- Entire software development is continued iteratively

- Different part of software is completed at different stages

- If deficiency is found go back to earlier stage if necessary to correct it

- Signs of missing object includes:

* Asymmetries in association and generalization
* Disparate attributes and operation on a class
* Signs of unnecessary class includes:
* Lacks of attributes, operations and associations on class
* Signs of missing association includes:
* Missing access path for operation: add new association so that queries can be answered
* Signs of unnecessary association includes:
* Redundant information in association
* Signs of incorrect placement of association
* Rolenames are too broad or too narrow

**8.2.11 Grouping classes into modules**

- Last step of object modeling is to group classes into sheets and modules

- Diagrams may be divided into sheets of uniform size for convenience in drawing, printing

- Tightly coupled classes should be grouped together

- Module is a set of classes that captures some logical subset of entire model

- Modules may vary in size

- A star pattern is useful for organizing modules

- Modules may be reused from previous design

- Reuse is easiest when a part of problem domain matches a previous problem

- If the new problem is different, the original design may have to be extended to encompass both problems

- A single module contains top level structure of high level class

- Other modules expand each high level class into generalization hierarchy and add associations to additional low-level classes

**8.3 Dynamic Modeling**

- Dynamic model shows time dependent behavior of the system and object in it

- Dynamic model summarizes event sequences for each object with a state diagram

- Dynamic model is important for interactive systems

- Steps in constructing dynamic model are:

* Prepare scenario of typical interaction sequences
* Identify events between objects
* Prepare an event trace for each scenario
* Build a state diagram
* Match events between objects to verify consistency

**8.3.1 Preparing a scenario**

- Prepare one or more typical dialogs between user and system

- Scenarios show major interactions, external display format and information exchanges

- First prepare scenarios of normal cases, interaction without any unusual inputs or error condition

- Then consider special cases such as omitted input sequences, maximum and minimum values and repeated values

- Then consider user error cases including invalid values and failure to respond

- For many interactive application error handling is the most difficult part of implementation

- Consider various kinds of interactions such as help requests and status queries

- 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, sensor or another task.

|  |
| --- |
| ATM asks the user to insert card; the user inserts card  ATM accepts card and read its serial number  ATM requests password; user enters “1234”  ATM verifies password with consortium  ATM asks user to select the kind of transaction (withdrawal, deposit, ministatement); user selects withdrawal  ATM asks for the amount of cash; the user enters $100  ATM verifies that amount in account balance  ATM dispenses cash and asks user to take it; the user takes the cash  ATM asks whether the user wants to continue; user indicates no  ATM prints a receipt, ejects the card and asks user to take them; the user takes the receipt and the card  ATM asks user to insert card |

**Fig 8.1 Normal ATM Scenario**

- Fig 8.1 shows a normal ATM scenario

- Information values exchanged are parameters of event

- For each event identify the actor that caused the event and parameter of event

- Fig 8.2 shows a scenario with exceptions

- Other variations with exceptions are:

* User fails to insert card
* ATM can’t read card
* The card has expired
* The amount is invalid
* The machine is out of cash

|  |
| --- |
| ATM asks the user to insert card; the user inserts card  ATM accepts card and read its serial number  ATM requests password; user enters “9999”  ATM verifies password with the consortium  ATM indicates incorrect password and asks user to renter it; the user enters “1234” which the ATM verifies successfully with consortium  ATM asks user to select the kind of transaction (withdrawal, deposit, ministatement); user selects withdrawal  ATM asks for the amount of cash; the user has a change of mind and hits cancel  ATM ejects the card and asks user to take it; the user takes it  ATM asks user to insert card |

**Fig 8.2 ATM Scenario with exception**

**8.3.2 Interface formats**

- Most interactions can be separated into two parts:

* Application logic
* User interface
* The analysis should concentrate first on the information flow and control rather than presentation format
* Dynamic model captures the control logic of application
* It is hard to evaluate a user interface without testing it
* Often the interface can be mocked up so that users can try it
* Application logic can often be simulated with dummy procedures

**8.3.3 Identifying events**

- Examine the scenarios to identify all external events

- Events include inputs, decisions, interrupts, transitions and actions to or from users or - Use scenarios to find normal events

- An action by an object that transmits information is an event

- Eg: Enter password is an event sent from external agent user to application object ATM

insert card

enter password

enter amount

take cash, take card

cancel,continue

display main screen

unreadable card message

request password

request kind , amount

canceled message

eject card,dispense cash

print receipt,

verify card

process bank transaction

bank account OK

bank transaction succeed

**Fig 8.3 Event flow diagram for ATM**

User

Consortium

verify account transaction succeed,failed

process transaction account ok

Bank

ATM

verify card

Bank transaction succeed,failed

Bank account OK

**8.3.4 Building a state diagram**

- Prepare a state diagram for each object class showing events the object receives and sends

- Every scenario or event trace corresponds to a path through the state diagram

- Start with event trace diagram

- Pick a trace showing typical interaction and consider events affecting a single object

- Arrange events into path where arcs are labeled by input and output events

- The interval between any two events is a state

- Give each state a name

- The initial diagram will be sequence of events and state

- Find loops within the diagram

- If a sequence of events can be repeated indefinitely then they form a loop

- Replace finite sequence of events with loops when possible

- Merge other scenarios into state diagram

- While examining states and scenarios, possible events can occur. Add them to state diagram

- After normal events have been considered, add boundary cases and special cases

- Consider events that occur at awkward times

- Eg: A request to cancel a transaction after it has been submitted for processing

- State diagram is finished when diagram covers all scenarios and diagram handles all events that can affect an object of class

- If there are complex interactions use nested state diagram

User ATM Consortium Bank

insert card

request password

enter password

Transaction suceed

process transaction

Process bank transaction

Bank transaction suceed

Dispense cash

account OK

verify account

verify card

bank account OK

request kind

enter kind

request amount

enter amount

terminate

Print receipt

Eject card

Take card

Display main screen

Request take cash

Take cash

**Fig 8.4 Event Trace for ATM Scenario**

**8.3.5 Matching events between objects**

- Check for completeness and consistency at system level when state diagram for each class are completed

- Every event should have a sender and receiver

- State should represent starting or termination point

- Make sure that events on different state diagram are consistent

**8.4 Functional Modeling**

- Functional Modeling shows how values are computed

- Functional modeling shows which values depend on which other values and the functions that relate them

- Data flow diagram are useful for showing functional dependencies

- Process transforms data

- Data flow connects output of one process to input of another process

**- Steps for constructing a functional model:**

* Identify input and output
* Build data flow diagram showing functional dependencies
* Describe functions
* Identify constraints
* Specify optimization criteria

**8.4.1 Identifying input and output values**

- List input and output values

- Examine the problem statement to find any input or output values that missed

**8.4.2 Building data flow diagram**

- Construct a data flow diagram showing how each output value is computed from input values

- DFD is usually constructed in layers

- The top layer consist of a single process, values and generate output

- Output values are supplied and consumed by external objects

- Expand each process into lower level data flow diagram

- Data flow diagram specify only dependencies among operations

- They do not show decisions or sequencing of operation

- Decisions do not directly affect output values in data flow diagram

- The output of decision functions can be shown as control signals indicated by dotted output arrows

- Eg: verify password is a decision function

Administrator

Login file

Enter user

Rejected

Ok

User name Password

**Fig 8.5 DFD for Login**

**8.4.3 Describing function**

- When the data flow diagram has been refined write a description of each function

- The description can be natural language, mathematical function, pseudocode, decision tables

- Focus on what the function does, not how to implement it

- The description can be declarative or procedural

- Declarative description specifies the relationship between input and output values and relationship among output values

- Procedural description specifies a function by giving an algorithm to compute it

- The purpose of algorithm is to specify what the function does

- Declarative descriptions are preferable to procedural description because they do not imply an implementation

**8.4.4 Identifying constraints between objects**

- Constraints are functional dependencies between objects that are not related by an input output dependency

- Constraints can be on two objects at same time; between instances of same object at different times or between instances of different object at different times

- Preconditions on functions are constraints that the input values must satisfy

- Postconditions are constraints that the output values are guaranteed to hold

**8.4.5 Specifying optimization criteria**

- Specify values to be maximized, minimized or otherwise optimized

- Eg: Optimization critieria for ATM example include: Minimize the number of physical messages sent between different sites. Minimize the time an account is locked for concurrency reasons

**8.5 Adding Operations**

- The list of useful operations is open ended and it is difficult to know when to stop adding them

- It is useful to distinguish various kinds of operations during analysis

- Key operations should be summarized in object model

**8.5.1 Operations from object model**

- Operations from object structure include reading and writing attribute values and association links

- Dot notation indicate attribute access

- Eg: ATM . cash-on-hand

- Navigating a path from one object to another through object model can be expressed as a series of pseudo-attribute Eg: account . bank

- Accessing a qualified link can be shown using index notation

**8.5.2 Operations from events**

- Each event sent to an object correspond to an operation on the object

- Events can be implemented directly by including an event handler

- During analysis events are best represented as labels on state transitions

**8.5.3 Operations from state actions and activities**

- Actions and activities in state diagram may be functions

- These functions having computational structure should be defined as operations

- Eg: In ATM example Bank has activity verify password

**8.5.4 Operations from functions**

- Each function in DFD corresponds to an operation on an object

- These functions have interesting computational structure and should be summarized on object model

- Organize the functions into operations on objects

**8.5.5 Shopping list operation**

- Shopping list means operations are not dependent on particular application nor subject to a particular order of execution but are meaningful in their own right

- Shopping list operations permit future possible needs

**8.5.6 Simplifying operations**

- Examine the object model for similar operation

- Use inheritance where possible to reduce the number of operations

- Introduce new superclasses to simplify operations

- Locate each operation at correct level within class hierarchy

**8.6 Iterating the analysis**

- Most analysis model require more than one pass to complete. To understand a problem analyze iteratively. Verify the final analysis with the requestor and application domain experts

**8.6.1 Refining the analysis model**

- The overall analysis model may show inconsistencies within and across the model

- Iterate the different stages to produce a cleaner and more coherent design

- Refine object definitions to increase sharing and improve structure

- Add the details obtained during first pass

- Sometimes major restructuring in model is needed

- Remove objects or associations that seems useful at first.

- A good model do not have extra detail

**8.6.2 Restating the requirements**

- When the analysis is complete the model serves as basis for the requirements and defines the scope of future

- Some requirements specify performance constraints and others the method of solution

- The final model should be verified with the requestor

- During analysis some requirements may appear to be incorrect or impractical; correction to the requirements should be confirmed

- The analysis model should be verified by application domain experts to make sure that it correctly models the real world

- The final verified analysis model serves as basis for system architecture, design and implementation

**8.6.3 Analysis and design**

- The goal of analysis is to fully specify the problem

- The purpose of rules is to preserve flexibility and permit changes later

**CHAPTER-9**

**System Design**

* System design is for solving the problem and building a solution
* System design includes decisions about the organization of system into subsystem, the allocation of subsystem to hardware and software components and policy decisions that form framework for detailed design
* The overall organization of a system is called system architecture

**9.1 Overview of system design**

- During analysis the focus is on what needs to be done, independent of how it is done

- During design, decisions are made about how the problem will be solved first at high level then at detailed levels

- System design is the first stage in which basic approach to solve the problem is selected

- The system design must take the following decisions:

* Organize the system into subsystem
* Identify concurrency inherent in the problem
* Allocate subsystem to processors and tasks
* Choose an approach for management of data stores
* Handle access to global resources
* Choose implementation of control in software
* Handle boundary condition
* Set trade-off priorities

**9.1.1 Breaking system into subsystem**

- Divide the system into small number of components

- Each major component of a system is called subsystem

- Subsystem is package of classes, associations, operations and constraints

- Subsystem is identified by the service it provides

- Service is a group of related functions that share common purpose

- Each subsystem has a well defined interface

- A system should be divided into small number of subsystems

- The lowest level subsystems are called modules

- The relationship between two subsystems can be client-supplier or peer-to-peer

- In client-supplier relationship the client calls on the supplier which performs some service and replies with the result

- The client must know the interface of supplier but the supplier does not know the interface of client because all interactions are initiated by clients using supplier interface

- In peer-to-peer relationship each of the subsystem may call on others

- Peer to peer interactions are more complicated because the subsystem must know each others interface

- The decomposition of systems into subsystems may be organized as a sequence of horizontal layers or vertical partitions

**9.1.1.1 Layers**

- A layered system is an ordered set of virtual world, each built in terms of ones below it and providing the basis of implementation for ones above it

- The objects in each layer can be independent

- A subsystem knows about the layer below it, but has no knowledge of layer above it

- Layered architecture come in two forms: open and closed

- In a closed architecture each layer is built in terms of immediate lower layer

- In open architecture a layer can use features of any lower layer to any depth

- This open architecture reduces the need to redefine operations at each level which can result in a more efficient and compact code

**9.1.1.2 Partitions**

- Partitions vertically divide a system into several independent or weakly coupled subsystem

- A system can be decomposed into subsystems using both layers and partitons

- Layers can be partitioned and partitions can be layered

- Fig 9.1 shows a block diagram of typical application which involves simulation of application and interactive graphics

|  |
| --- |
| Application package |
| Simulation  package  screen graphics  pixel graphics  window graphics  User  Dialog  control |
| operating system |
| computer hardware |

**Fig 9.1 Block Diagram for Typical Application**

**9.1.1.3 System Topology**

- When the top-level subsystems are identified the designer should show information flow among subsystem with data flow diagram

**Fig 9.1 Block diagram of a typical application**

- Use simple topologies when they are suitable for a problem to reduce the number of interactions among subsystems

**9.1.2 Identifying concurrency**

- In analysis model all objects are concurrent

- In implementation not all objects are concurrent, because one processor may support many objects

- One important goal of system design is to identify which objects must be active concurrently

**9.1.2.1 Identifying inherent concurrency**

- Identify concurrency

- Two objects are inherently concurrent if they can receive events at the same time without interacting

- If the events are unsynchronized the objects cannot be folded onto a single thread of control

- Eg: The engine and wing controls on airplane must operate concurrently

- Two subsystems that are inherently concurrent need not to be implemented as separate hardware units. The purpose of hardware interrupts, operating system and tasking mechanism is to simulate logical concurrency in a uniprocessor

- If there are no timing constraints multitasking operating system can handle the computation

**9.1.2.2 Defining concurrent tasks**

- Although all objects are concurrent many objects are interdependent

- By examining state diagram of individual objects and the exchange of events among them many objects can be folded together on single thread of control

- A thread of control is a path through a set of state diagram on which a single object at a time is active

- A thread remains within a state diagram until an object sends an event to another object and waits for another event

- The thread splits if the object sends an event and continues executing

- On each thread of control only a single object at a time is active

- Thread of control are implemented as tasks in computer system

- Eg: While the bank is verifying an account or processing a bank transaction the ATM machine is idle. If the ATM is directly controlled by a central computer, then the ATM object can be folded together with the bank transaction object as a single task

**9.1.3 Allocating subsystems to processors and tasks**

- Each concurrent subsystem must be allocated to a hardware unit

- The system designer must:

* Estimate performance needs and the resources needed to satisfy them
* Choose hardware or software implementation for subsystem
* Allocate software subsystems to processors to satisfy performance needs and minimize interprocess communication
* Determine the connectivity of physical units that implement the subsystem

**9.1.3.1 Estimating hardware resource requirements**

- Multiple processors or hardware functional units is based on need for higher

performance than single CPU

* The number of processors required depends on the volume of computation and speed of machine
* The system designer must estimate the required CPU processing power by computing the steady state load as the product of number of transactions per second and the time required to process a transaction
* The amount of excess capacity needed depends on the acceptable rate of failure due to insufficient resources

**9.1.3.2 Hardware-Software trade offs**

- Hardware can be regarded as a rigid but highly optimized form of software

- The system designer must decide which subsystem will be implemented in hardware

and which in software

* Subsystems are implemented in hardware for two main reasons:
* Existing hardware provides the functionality required
* High performance

**9.1.3.3 Allocating tasks to processors**

- The tasks for various software subsystems must be allocated to processors

- Tasks are assigned to processors because

* Certain tasks are required to control hardware or to permit independent or concurrent operation
* The response time of information flow rate exceeds available communication bandwidth between task and piece of hardware
* Computation rates are too great for single processor, so tasks must be spread among several processors

**9.1.3.4 Determining physical connectivity**

- After determining the kinds and numbers of physical unit, the system designer must choose the arrangement and form connection among physical units

- The decisions that must be made are:

* Choose the topology for connecting the physical unit. Associations in object model correspond to physical connection. Some connections may be indirect but the designer must minimize the connection cost
* Choose topology of repeated units. If several copies of particular kind of unit or group of units are included for performance reasons, their topology must be specified
* Choose the form of connection channels and communication protocols. The system design phase specify the exact interfaces among units but interaction mechanisms and protocols must be chosen

**9.1.4 Management of data stores**

- Data store combine data structures, files and databases implemented in memory or secondary storage devices

- Files are cheap, simple and permanent form of datastore

- File implementations vary for different computer systems

- Databases managed by database management systems are another kind of data store

- Disadvantage of DBMS is that DBMS has a complex interface

- Guidelines that characterize the kind of data that belongs in formal database:

* Data that requires access at fine levels of detail by multiple users
* Data that can be effectively managed with DBMS commands
* Data that must port across many hardware and operating system platform
* Data that must be accessible by more than one application program
* Guidelines that characterize the kind of data that belongs in a file
* Data that is voluminous in quantity but difficult to structure
* Data that is voluminous in quantity and of low information density
* Raw data that is summarized in the database
* Volatile data that is kept a short time and then discarded

**9.1.4.1 Advantages of using a database**

- Many infrastructure features such as crash recovery, sharing between multiple users, sharing between multiple application, data distribution, integrity have

been programmed by DBMS vendor

* Common interface for all applications
* A standard access language:- The SQL language is supported by relational database management system

**9.1.4.2 Disadvantages of using a database**

- Performance overhead:- few relational database management system can exceed 50 simple transactions per sec on a computer

- Insufficient functionality for advanced application:- Relational DBMS were developed for business application that have large quantities of data with simple structure. Relational DBMS are difficult to use for applications that require richer data types

- Awkward interface with programming language:- Relational DBMS support set oriented operations that are expressed through non procedural language. Most programming languages are procedural in nature

**9.1.5 Handling global resources**

- The system designer must identify global resources and determine mechanisms for controlling access to them

- Global resources include physical units such as processors, tape drives, satellite; space such as disk space and button on a mouse

- If the resource is a physical unit then it can control itself by establishing a protocol for obtaining access within a concurrent system

**-** If the resource is a logical entity such as object ID or a database then there is a danger of conflicting access in shared environment

- Each global resource must be owned by a guardian object that controls access to it

- One guardian object can control several resources

- All access to resources must pass through the guardian object

- Allocating each shared global resource to a single object is a recognition that the resource has identity

- In a time-critical application, the cost of passing all access to a resource through a guardian object is sometimes too high and clients must access the resources directly.

- In this case locks can be placed on subsets of resource

- Lock is a logical object associated with some resource that gives the lock holder the right to access the resource directly

- Guardian object must exist to allocate the locks but after one interaction with the guardian to obtain a lock the user of the resource can access the resource directly

**9.1.6 Choosing software control implementation**

- During analysis all interactions are shown as events between objects

- Hardware control matches the analysis model; but system designer must choose several ways to implement control in software

- There are two kinds of control flow: External control and internal control

- External control is the flow of externally visible events among objects

- There are three kinds of external events: procedure driven sequential, event-driven sequential system and concurrent system

- Internal control is the flow of control within a process

**9.1.6.1 Procedure driven system**

- Control resides within the program code

- Procedures issues request for external input and wait for it. When the input arrives, control resumes within the procedure

- Advantage: easy to implement

- Disadvanatge: Requires concurrency inherent in objects to be mapped into sequential flow of control, the designer must convert events into operations between objects

**9.1.6.2 Event driven systems**

- Control resides within a dispatcher or monitor provided by the language or operating system

- Application procedures are attached to events and are called by dispatcher when corresponding events occurs

- Events are handled directly by the dispatcher

- Procedures must use global variables to maintain state or dispatcher must maintain local states

- Event driven control is more difficult to implement than procedure driven control

- Advantage: useful, simpler, modular and handle error conditions

**9.1.6.3 Concurrent systems**

- Control resides concurrently in several independent objects

- Events are implemented directly as one-way messages

- A task can wait for input, but other tasks continue execution

- Operating system supplies a queuing mechanism for events so that events are not lost if a task is executing when they arrive

- Operating system resolves scheduling conflicts among tasks

- Eg:tasks on an operating system

- If there are multiple CPUs then difficult tasks can execute concurrently

**9.1.6.4 Internal control**

- During design process operations on objects are expanded into lower level operations

- Internal interactions between objects can be viewed to external interactions among objects as events passed between objects

**9.1.6.5 Other paradigms**

- Rule based systems, logic programming systems and other forms of non procedural programs

- These languages are different from procedural languages

**9.1.7 Handling boundary conditions**

- System designer must consider boundary such as:

* **Initialization**:- In initialization the system must be brought from initial state to steady state condition. Things to be initialized include parameters, global variables, tasks. During initialization only a subset of functionality of system is available. Initializing a system containing concurrent tasks is difficult
* **Termination:**- Termination is simpler than initialization because many internal objects can be abandoned
* **Failure**:- Failure is the unplanned termination of a system. Failure can arise from user errors or from bugs in the system

**9.1.8 Setting trade-off priorities**

- The system designer must set priorities that will be used to guide trade-offs during test of design

- Design trade-offs must be made regarding the process of developing it

- The system designer must determine importance of criteria as a guide to make trade-off decisions during design

- All trade-offs are not made during system design, but the priorities for making them are established

**9.1.9 Common architectural frameworks**

- The common architectural frameworks are:

* Batch transformation
* Continuous transformation
* Interactive interface
* Dynamic simulation
* Real-time system
* Transaction manager

**9.1.9.1 Batch transformation**

- Sequential input to output transformation

- Inputs are supplied at start and goal is to compute an answer

- Most important aspect of batch transformation is functional model which specifies how input values are transformed into output values

- Eg: Compiler

- Steps in batch transformation:

* Break the overall transformation into stages, each stage performing one part of transformation. The system diagram is a dataflow diagram
* Define intermediate object classes for data flows between each pair of successive stages
* Each stage knows only its own input and output
* Expand each stage until the operation can be implemented
* Restructure the final pipeline

parse tree

token list

source text

optimized code sequence

augemented

Parse tree

Code sequence

object code

**Fig 9.2 Data flow diagram of compiler**

**9.1.9.2 Continuous Transformation**

- System in which outputs actively depend on changing input

- The outputs must be updated frequently

- The functional model together with object model defines the values being computed

- Dynamic model is less important

- Because of severe time constraints the entire set of outputs cannot be recomputed

- So the new output values must be computed incrementally

- To make incremental computation intermediate objects may be defined to hold intermediate values

Window Position

Window Bounds

Application Coordinate System

Clipped Image

Viewspace Image

Screen Image

Graphic Model

**Fig 9.3 Data flow diagram for a graphics application**

- Steps in designing a pipeline for continuous transformation are:

* Draw a data flow diagram for the system. The input and output actors correspond to data structures whose value change continuously
* Define intermediate objects between each pair of successive stages
* Differentiate each operation to obtain incremental changes to each stage
* Add additional intermediate objects for optimization

**9.1.9.3 Interactive interface**

- System that is dominated by interactions between the system and external agents such as humans, devices or other programs

- Interactive interface includes only part of an entire application

- Dominated by dynamic model

- Eg: Control panel for simulation, a workstation windowing system

- Interactive system is concerned with external appearances

- Steps in designing an interactive interface:

* Isolate the objects that form the interface from the object that define the semantics of the application
* Use predefined objects to interact with external agents
* Use dynamic model as the structure of program
* Isolate physical events from logical events
* Fully specify the application function that are invoked by the interface. Make sure that the information to implement them is present

**9.1.9.4 Dynamic simulation**

- Models or tracks objects in real world Eg: Video games

- Steps:

* Identify actors, active real world objects from object model. The actors have attributes that are periodically updated
* Identify discrete events. Discrete events can be implemented as operations
* Identify continuous dependencies. Real world attributes may be dependent on other real world attributes or vary continuously with time, velocity
* Discrete events can be exchanged as a part of timing loop

**9.1.9.5 Real time system**

- Interactive system for which time constraints are tight or slightest timing failure cannot be tolerated

- To guarantee response time the worst case scenario has to be determined

- Complex

- Involves issues such as interrupt handling and coordinating multiple tasks

**9.1.9.6 Transaction manager**

- Database system whose main function is to store and access information

- The information comes from application domain

- Most transaction managers must deal with multiple users and concurrency

- Eg: airline reservation system

- The object model is important

- Functional model is less important

- Dynamic model is important for estimating transaction throughput

- Steps:

* Map the object model directly into database
* Determine units of concurrency. Introduce new classes as needed
* Determine the unit of transaction
* Design concurrency control for transaction