**Module-4**

**Chapter-10**

**Object Design**

* The object design phase determines the full definition of classes and associations used in implementation and interface
* Object design phase adds internal objects for implementation and optimizes data structure and algorithms

**10.1 Steps in object design**

- Combine the three models to obtain operations on classes

- Design algorithms to implement operation

- Optimize access paths to data

- Implement control for external interactions

- Adjust class structure to increase inheritance

- Design associations

- Determine object representation

- Package classes and associations into modules

**10.1.1 Combine the three models**

- After analysis we have object, dynamic and functional model, but object model is the main framework around which the design is constructed

- The object model from analysis may not show operations

- The designer must convert action and activities of dynamic model and process of functional model into operations

- State diagram describes the life history of an object

- **Transition** is change of state of object

- In the state diagram the action performed by **transition** depends on both events and state of object

- The algorithm implementing an operation depends on state of object

- Event sent by object may represent an operation on another object

- Events occur in pairs with first event triggering an action and second event returning result or indicating completion of that action

- An action or activity initiated by a transition in state diagram may expand into entire data flow diagram

- The designer must convert the structure of diagram into linear sequence of steps in an algorithm

- The processes in data flow diagram constitute sub operations

- Determine the target object as follows:

* If a process extracts a value from an input flow then the input flow is the target
* If a process has an input flow and output flow of same type and output value is an updating version of input flow then the input or output flow is the target
* If a process has an input from or output to a data store or an actor, then data store or actor is a target of process

**10.1.2 Designing algorithm**

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

- The analysis tells what the operation does and algorithm tells how it is done

- An algorithm may be subdivided into simple operation

- The algorithm designer must:

* Choose algorithm that minimize the cost of implementing operations
* Select data structure appropriate to algorithms
* Define new internal classes and operations as necessary
* Assign responsibility for operations to appropriate class

**10.1.2.1 Choosing algorithms**

-Many operations are simple enough that specification in functional model constitutes

a satisfactory algorithm because the description of what is done also shows how it is

done

* Eg: Diagram class box object containing an operation list contains a set operation entry objects. There is no need to write an algorithm to find class box containing a given operation entry object
* Some functions are specified as declarative constraints without any procedural definition i.e we have to use our knowledge to invent an algorithm
* Most functions have simple mathematical or procedural definition
* Eg: Class box is drawn by first drawing its outline anf then drawing its parts, the operation list and attribute list

list

box

Operation entry

Class box

Operation list

entry

operation

box

list

attributes

entry

Attribute entry

Attribute list

**Fig 10.1 Fragment of OMTool Model**

-The level of abstraction of algorithms should not go below the level of granularity of objects

- It is not necessary to write algorithms for operation that are internal to one object

- Considerations in choosing alternative algorithms include:

* Computational complexity: How execution time grows with number of input values such as constant time as well as cost of processing each input value
* Ease of implementation and understandability: Provide operations that can be implemented quickly with a simple algorithm
* Flexibility: Check whether the algorithm is readable and correct
* Fine tuning the object model: If the object model were structured differently there would be other alternatives. In original upper design each diagram contains a list of windows in which it is visible. This is insufficient because operations on set of elements must be computed separately for each window

visible

Window

Diagram

Window

Sheet

Diagram element

visible

sheet

**Fig 10.2 Alternative structure**

**10.1.2.2 Choosing data structure**

- Choosing algorithm involves choosing the data structure

- During analysis we concentrated on logical structure of information, but during object design we must choose the form of data structure that will permit efficient algorithm

- Data structure do not add information to analysis model but they organize it in a form convenient for algorithms that use it

- Implementation data structures are instances of container classes

* Data structures include arrays, queues, stacks and trees

**10.1.2.3 Defining internal classes and operations**

-During expansion of algorithms new classes may be needed to hold intermediate

results

* New low level operations may be invented during decomposition of high level operations
* The low level operations may be defined during object design because most of them are not visible externally
* Eg: In OMTool erase operation to erase an object it must be drawn in background color, then objects uncovered by erasure or damaged by the draw must be repaired

by being redrawn

**10.1.2.4 Assigning responsibility for operations**

- For each operation some responsibilities are assigned

- Those operations can be performed at several places in an algorithm

- It is difficult when more object is involved in operation

- Ask the following questions:

* Is one object modified by operation while other objects are only queried for information they contain. The object that is changed is the target of operation
* Looking at the classes and associations that are involved in operation, which class is located centrally in sub network of object model? If classes and association form a star, it is the target of operation
* If one object acted on while the other object performs the action?

**10.1.3 Design Optimization**

- The basic design model uses the analysis model as framework for implementation

- Analysis model captures logical information about the system

- Design model adds details to support efficient information access

- During design optimization the designer must:

* Add redundant associations to minimize access cost and maximize convenience
* Rearrange the computation for greater efficiency
* Save derived attributes to avoid recomputation of complicated expression

**10.1.3.1 Adding redundant association for efficient access**

- During analysis redundant association do not add any information

- During design we evaluate the structure of object model for implementation

Has-skill

Employs

Skill

Person

Company

**Fig 10.3 Chain of associations**

**-** Suppose the company has 1000 employees each of whom has 10 skills. Simple

nested loop would traverse Employs 1000 times and Has-skill 10,000 times. If only

5 employees actually speak Japanese then test to hit ratio is 2000.

**-** When the number of hit from a query is low because only a fraction of objects

satisfy the test we can build an index to improve access to objects that must be

frequently retrieved

- Eg: We can add a qualified association Speaks language from Company to

Employee where the qualifier is the language spoken

- There is a cost of index: It requires additional memory and it must be updated

whenever base associations are updated

Speaks language

Person

langauge

Company

**Fig 10.4 Index for personal skill database**

- Speaks language is a derived association

- Derived association does not add any information to the network

**-** Analyze the use of paths in association network as follows:

* Examine each operation and see what associations it must traverse to obtain information. Note which associations are traversed in both directions and which are traversed in single direction

- For each operation note the following items

* How often is the operation called? How costly is it to perform?
* What is the fan-out along a path through the network?
* What is the fraction of hits on the final class that is objects that meet selection criteria?

**10.1.3.2 Rearranging execution order for efficiency**

- Optimize the algorithm after adjusting the structure of object model

- Data structures and algorithms are related but data structures should be considered

first

* One key to algorithm is to eliminate dead paths(can never be accessed) as early as possible
* Eg: Suppose we want to find all employees who speak both Japanese and French. Suppose 5 employees speak Japanese and 100 speak French, it is better to test and find Japanese speakers first, then test if they speak French

**10.1.3.3 Saving derived attributes to avoid recomputation**

- Data that is redundant because it can be derived from other data can be cached or

stored in computed form to avoid recomputation

- New object or classes may be defined to retain information

- The classes that contains cached data must be updated if any objects that it depends

on are changed

- Fig 10.5 shows use of derived object and derived attribute

- Each class contains an ordered list of attributes and operations each represented as

text string. Given the location of class box; the location of each attribute can be

computed by adding the size of all elements. Since the location of each element is

needed frequently, the location of each attribute is computed and stored. The region

containing entire attribute list is computed and saved so that input points need not be

tested

**Attribute**

text/location

**Attributelist**

region

{ordered}

**Class box**

location

**Attribute**

text

**Operation**

text

{ordered}

**Class box**

location

**Operation**

text/location

**Operation list**

region

{ordered}

{ordered}

**Fig 10.5 Derived attribute to avoid recomputation**

- These datas contain attributes. These attributes are called derived attributes

- The use of association as cache is shown in fig 10.6

- A sheet contains a priority list of partially overlapping elements. If an element is

moved or deleted the elements under it must be redrawn. Overlapping elements can

be found by scanning all elements in front of deleted element in the priority list for

sheet and comparing them to deleted element. Overlaps association stores elements

that overlap an object and precede it in list. This association must be updated when

a new element is added

Priority list

next

over

**Diagram element**

Previous

Overlaps

under

**Fig 10.6 Association as a cache**

- Derived attributes must be updated when base value changes

- There are three ways to recognize when an update is needed

* **Explicit update**: Each derived attribute is defined in terms of one or more fundamental base object. The designer determines which derived attributes are affected by each changes to fundamental attribute and inserts code into update operation on base object to update the derived attributes that depend on it
* **Periodic recomputation**: Base values are often updated in bunches. Recomputation of derived attributes can be more efficient than incremental update because some derived attribute may depend on several base attribute and might be updated more than once
* **Active values**: Value that has dependent values are called active values. Each dependent value registers itself with active value which contains a set of dependent values and update operation. Some programming language implement active values

**10.1.4 Implementation of control**

- The designer must refine the strategy for implementing the state event present in dynamic model

- There are three basic approaches to implement the dynamic model

* Using the location within the program to hold state (procedure driven system)
* Direct implementation of state machine mechanism (event driven system)
* Using concurrent tasks

**10.1.4.1 State as location within a program**

- Traditional approach to represent control within a program

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

- Any finite state machine can be implemented as a program

- After input is read the program branches depending on input event received

- Disadvantage: Lack of modularity

- One technique to convert a state diagram to code is as follows:

* Identify the main control path. Beginning with initial state identify a path through the diagram that corresponds to normally expected sequence of events. Write the name of states along this path as a linear sequence
* Identify alternate paths that branch off the main path and rejoin it later. These become conditional statement in the program
* Identify backward path that branch off the main loop and rejoin it earlier. These become loops in program. If there are multiple backward paths that do not cross, they become nested loops in program
* The state and transitions that remain correspond to exception conditions. They can be handled by several techniques including exception handling supported by language

**pseudocode for atm**

do forever

display main screen

read card

repeat

ask for password

read password

verify account

until account verification is ok

repeat

ask for kind of transaction

read kind

ask for amount

read amount

start transaction

wait for it to complete

until transaction is ok

dispense cash

wait for customer to take it

ask whether to continue

until user asks to terminate

eject card

wait for customer to take card

Main screen

insert card

Request password

enter password

Verify account

bank account

Account OK

Request kind

Request amount

enter amount

Process transaction

transaction failed

transaction succeed

enter kind

Dispense cash

take cash

Request continuation

continue

terminate

finish

Card ejected

take card

**Fig 10.7 ATM Control**

**10.1.4.2 State machine engine**

- Direct approach to implement control is to have some way of explicitly representing and executing state machine

- State machine engine class provides the capability to execute a state machine represented by a table of transitions and actions provided by application

- Each object instance has its own state variables but would call on state machine to determine the next state and action

- State machines are objects but not application objects

- They are part of language substrate to support semantics of application objects

- This approach allows to quickly progress from analysis model to prototype of system by defining classes from object model, state machines from dynamic model and creating stubs

- Stub is a minimal definition of function and subroutine without any internal code

- A parser such as Unix yacc or lex produce an explicit state machine to implement a user interface

**10.1.4.3 Control as concurrent tasks**

**-** An object can be implemented as a task in programming language or operating system

- This is the most general approach as it preserves the inherent concurrency of real objects

- Events are implemented as inter task calls using the facilities of language or operating system

- Some languages such as concurrent Pascal or concurrent C++ supports concurrency

- Ada supports concurrency but its run time cost is high

**10.1.5 Adjustment of inheritance**

- The definition of classes and operations can be adjusted to increase the amount of inheritance. The designer should

* Rearrange and adjust classes and operations to increase inheritance
* Abstract common behavior out of group of classes
* Use delegation to share behavior when inheritance is semantically invalid

**10.1.5.1 Rearranging classes and operations**

` - Sometimes same operation is defined across several classes and can be easily be inherited from common ancestor

- Some operations are not identical

- Before inheritance can be used each operation must have same interface same semantics and same signature

- Same signature means same number and type of arguments and results

- If the signature match then the operations must be examined to see if they have same semantics

- The following kind of adjustments can be used to increase the chance of inheritance

* Some operations may have fewer arguments than others. The missing arguments can be added or ignored
* Some operations may have fewer arguments because they are special cases of general arguments. Implement the special operation by calling the genral operation with appropriate parameter values
* Similar attributes in different classes may have different names. Give the attributes the same name and move them to a common ancestor class. Also check similar operations with different names
* An operation may be defined on several different classes in group but may be undefined on other classes. Define it on common ancestor class and declare it as no operation on class that don’t care about it

**10.1.5.2 Abstracting out common behavior**

- When a common behavior has been recognized a common superclass can be created that implements shared features

- This transformation of object model is called abstracting out of common behavior

- Usually the resulting class is abstract means that there is no direct instance of it

- The abstract superclass created is reusable in future projects

- Advantage: Sharing, reusable, modularity and extensibility

**10.1.5.3 Use delegation to share implementation**

- Inheritance is a mechanism for implementing generalization in which behavior of superclass is shared by subclass

- Sharing of behavior is justifiable only when a true generalization relationship occurs

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

- One object can invoke the desired functions of another class using delegation rather than inheritance

- Delegation consist of catching an operation on one object and sending it to another object

- For eg: Every instance of Stack contains a private instance of List. Stack:push

operation delegates to list by calling last and add operation to add an element at the

end of list and pop operation has similar implementation using last and remove

operation. The ability to corrupt the stack by adding or removing arbitrary elements

is hidden from client of Stack class

**List**

add

remove

first

last

**Stack**

**List**

add

remove

first

last

body:list {private}

push

pop

**Stack**

push

pop

**Discouraged Recommended**

**Fig 10.8 Alternative implementation of stack using inheritance(left) and delegation(right)**

**10.1.6 Design of associations**

- Associations are glue of object model that provides access paths between objects

- Associations are conceptual entities useful for modeling and analysis

- During object design phase we must formulate a strategy for implementing association in object model

- Choose a global strategy for implementing association uniformly or select a particular technique for each association

**10.1.6.1 Analyzing association traversal**

- Associations are bidirectional

- If the associations are traversed in one direction their implementation can be simplified

- For prototype work we always use bidirectional associations so that we can add new behavior and expand or modify the application

- For production work we optimize some association

- Whichever implementation strategy you choose you should hide the implementation using access operation to traverse and update the association

**10.1.6.2 One way association**

- If an association is only traversed in one direction it may be implemented as a pointer- an attribute which contains an object reference

- If the multiplicity is one then it is simple pointer

- If the multiplicity is many then it is a set of pointers

- If the many end is ordered then a list can be used instead of set

- A qualified association with multiplicity one can be implemented as dictionary object

- Dictionary is a set of value pairs that maps selector values into target values

- Dictionaries are implemented efficiently in most object oriented languages using hashing

Works\_for

Company

Person

Company

Person

employee

employer

**Fig 10.9 Implementation of one way association using pointer**

**10.1.6.3 Two way associations**

- Many associations are traversed in both directions. There are three approaches to their implementation

* Implement as attribute in one direction only and perform a search when a backward traversal is required. This approach is useful only if there is a great disparity in traversal frequency in two directions and minimizing both storage cost and update cost are important. The rare backward traversal will be expensive
* Implement as attributes in both directions. This approach permits fast access. If either attribute is updated then the other attribute must be updated to keep the link consistent

Works\_for

Person

Company

Person

Company

employer

employees

Set

**Fig 10.10 Implementation of two-way association using pointer**

* Implement as distinct association object independent of either class. An association object is a set of pairs of associated objects stored in a single variable size object. For efficiency an association object can be implemented using two dictionary objects; one for forward direction and one for backward direction. This approach is useful for extending predefined class from a library which cannot be modified because the association object can be added without adding any attributes to original classes. Distinct association objects are useful for sparse association in which most objects of classes do not participate because space is used only for actual links

Works\_for

(**Company)**

**(Person)**

|  |  |
| --- | --- |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |

**(Person)**

**(Person)**

**(Person)**

**(Company)**

**(Person)**

**10.1.6.4 Link attributes**

- If the association has link attributes then its implementation depends on

multiplicity

- If the association is one to one the link attributes can be stored as attributes

of either object

- If the association is many to many the link attributes cannot be associated

with either object

**10.1.7 Object representation**

- Implementing object is straight forward but 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 integer, strings, and enumerated types

- Eg: Consider the implementation of social security number. The social security number attribute can be implemented as an integer or string or as an association to social security number object which can contain either integer or string

SS number

Employee

Employee

SSN:integer

SSN:integer

**10.1.8 Physical Packaging**

- Packaging involves the following issues:

* Hiding internal information from outside view
* Coherence of entities
* Constructing physical module

**10.1.8.1 Information hiding**

- Design goal is to treat classes as black boxes, whose external interface is public but whose internal details are hidden from view

- Hiding internal information permits implementation of class to be changed without requiring any client of class to modify code

- During analysis information hiding is not considered

- During design the public interface of each class must be carefully defined

- The designer must decide what attributes should be accessible from outside the class

- These decisions should be recorded in object model by adding the annotation {private} after attributes that are to be hidden or by separating the list of attributes into two parts

- During design scope of any one method is limited

- During design we define the bounds of visibility that each method requires

- Each operation should have a limited knowledge of entire model including the structure of classes associations and operations

- The design principles to limit the scope of knowledge of any operation are:

* Allocate to each class the responsibility of performing operations and providing information
* Call an operation to access attributes belonging to an object of another class
* Avoid traversing associations that are not connected to current class
* Define interfaces
* Hide external objects at system boundary by defining abstract interface classes
* Avoid applying a method to result of another method

**10.1.8.2 Coherence of Entities**

- One important design principle is coherence of entities

- An entity such as a class, operation, module is coherent if it is organized on consistent plan

- An entity should have a single major theme. It should not be collection of unrelated parts

- A single method should not contain both policy and implementation

- Policy is making of context dependent decisions

- Implementation is execution of fully specified algorithm

- Policy involves making decisions, gathering global information and interacting with outside world

- Policy method contains I/O statements, conditionals and access data stores

- Policy method does not contain complicated algorithm

- An implementation method does not make any decisions

- Separating policy and implementation increases the possibility of reusability

**10.1.8.3 Constructing modules**

- During analysis and system design the object model is partitioned into modules

- The new classes that are added during design can be organized into separate module or layer that did not exist in analysis

- Modules should be defined so that their interfaces are minimal and well defined

- The interface between two modules consist of association that relate classes in one module with other classes in other

- Modules should have some cohesiveness or unity of purpose

- The classes in a module should represent similar kind of things in application

- The connectivity of object model can be used for partitioning modules

- Classes that are closely connected should be in same module

- Classes that are not connected or loosely connected may be in separate modules

**10.1.9 Documenting design decisions**

- The design decisions should be documented when they are made

- Documentation is the best way of transmitting the design to others and recording it for reference during maintenance

- The design document should be an extension of Requirement Analysis Document

- Design document include detailed description of object model in both graphical form (object model diagram) & textual form (class descriptions)

- The functional model is extended during design phase

- If the dynamic model is implemented as an explicit state control or concurrent tasks then analysis model or its extension is adequate.

- If dynamic model is implemented by location within the program code then psuedocode for algorithm is needed

**10.2 Comparison of Methodology**

**-** Refer text Pg no:280-288