DESIGN AND IMPLEMENTATION OF AN OBJECT ORIENTED DATABASE MANAGEMENT SYSTEM KERNEL

A Master's Thesis

Presented by

Tansel OKAY

to

the Graduate School of Natural and Applied Sciences

of Middle East Technical University

in Partial Fulfillment for the Degree of

MASTER OF SCIENCE

in

COMPUTER ENGINEERING

MIDDLE EAST TECHNICAL UNIVERSITY

ANKARA

September, 1993

Approval of the Graduate School of Natural and Applied Sciences.

[illegible]

Prof. Dr. İsmail TOSUN
Director

I certify that this thesis satisfies all the requirements as a thesis for the degree of Master of Science.

[illegible]

Prof. Dr. Neşe YALABIK
Chairman of the Department

We certify that we have read this thesis and that in our opinion it is fully adequate, in scope and quality, as a thesis for the degree of Master of Science in Computer Engineering.

[illegible]

Prof. Dr. Asuman DOĞAÇ
Supervisor

Examining Committee in Charge:

Prof. Dr. Kemal İNAN [illegible]

Prof. Dr. Asuman DOĞAÇ [illegible]

Assoc. Prof. Dr. Sinan NEFTÇİ [illegible]

Assoc. Prof. Dr. Aral EGE [illegible]

Asst. Prof. Dr. Halit OĞUZTÜZÜN [illegible]

ABSTRACT

DESIGN AND IMPLEMENTATION OF AN OBJECT ORIENTED DATABASE MANAGEMENT SYSTEM KERNEL

OKAY, Tansel
M.S. in Computer Engineering
Supervisor: Prof. Dr. Asuman DOGAC

September, 1993, 115 pages

With the increasing ability of computers to store, transfer and process huge amounts of different kinds of information(i.e., image ,voice), software developers are forced to design and implement extensible database management systems which can operate on dynamically defined objects.

With this work, namely, MOOD Kernel, an extensible Object Oriented Database Management System Kernel, which can operate on dynamically defined multi-media objects as well as conventional textual information, is designed and implemented as a part of the MOOD(METU Object-oriented DBMS) project.

MOOD Kernel is based on Exodus Storage Manager(ESM). Therefore some of the kernel functions like concurrency control, crash recovery and storage management are readily provided by the ESM.

The Kernel functions implemented in this thesis, that supports dynamic schema modification, are catalog management and dynamic function definition and linking.

MOOD Kernel is integrated with MOODSQL,that is, the object-oriented SQL of MOOD, MOODVIEW, graphical user interface of MOOD and MOOD Algebra.

Keywords: Object-Oriented Databases, Database Kernel, Late Binding, Object Orientation, Catalog Management, Extensibility

Science Code : 619.02.02

ÖZ

NESNEYE YÖNELİK VERİ TABANI YÖNETİM SİSTEMİ ÇEKİRDEĞİ TASARIMI VE GERÇEKLEŞTİRİLMSİ

OKAY, Tansel

Yüksek Lisans Tezi, Bilgisayar Mühendisliği Anabilim Dalı

Tez yöneticisi: Prof. Dr. Asuman DOĞAÇ

Eylül, 1993, 115 sayfa

Bilgisayarların bilgi saklama, transfer etme ve işleme kapasitelerindeki hızlı artış yeni veri tiplerinin gündeme gelmesini (görüntü, ses) sağlamıştır.Böylece yazılım geliştirenler dinamik olarak tanımlanan nesneler üzerine çalışan, genişletilebilir veri tabanı tönetim sistemleri tasarlamak ve gerçekleştirmek ihtiyacını duymuşlardır.

Bu çalışmada , MOOD Çekirdek birimi, yani genişletilebilir bir nesneye yönelik veri tabanı yönetim çekirdeği geliştirilmiştir. Geliştirilen çekirdek, dinamik olarak tanımlanan veri yapıları(görüntü, ses) üzerinde çalişabildiği gibi önceden kullanılan karakter veriler üzerinde de çalışabilmektedir. Bu proje MOOD(ODTU nesneye yönelik veri tabanı yönetim sistemi)'nin bir parçası olarak geliştirilmiştir.

MOOD çekirdek birimi EXODUS(ESM) depolama yöneticisi kullanılarak gerçekleştirilmiştir, bunun için bazı veri tabanı işlemleri, örneğin eşzamanlılık kontrolu, hata düzeltici, depolama yönetimi ESM tarafından sağlanmaktadır.

Bu tezde gerçekleştirilen çekirdek fonksiyonları, dinamik yapı değişikliği, katalog yönetimi ve dinamik alt program tanımlanması ve bağlantısını sağlanmaktadır.

MOOD Çekirdek birimi, MOODSQL(nesneye yönelik sorgulama birimi), MOODVIEW(Kullanıcı ara birimi) ve MOOD Algebra birimleri ile entegre edilmiş ve çalışır durumdadır.

Anahtar Kelimeler: Nesneye yönelik veri tabanları, Veri tabanı çekirdeği, Sonradan bağlantı, Katalog Yönetimi, genişletilebilirlik.

Bilim dalı sayısal kodu: 619.02.02

ACKNOWLEDGEMENTS

I am grateful to many individuals for the cooperation, support and the encouragement they gave me while preparing this thesis. First of all, I would be honored to present my special thanks to my supervisor Prof. Dr. Asuman Dogac without whose help this work could have never been possible.

I am grateful to my friends, C.Ertan Ozkan, Isik Yigit, Sumer Gurkok, Cem Evrendilek, Ismailcem Budak Arpinar, Ismail Tore, Mehmet Altinel, Ilker Durusoy for their valuable comments on the work accomplished.

Finally, I want to thank to my family for their support and patience during this thesis.

TABLE OF CONTENTS

LIST OF FIGURES

page

CHAPTER I

INTRODUCTION

A database kernel has been designed and implemented as a part of MOOD(METU Object-Oriented DBMS) project.

The research on object-oriented DBMSs is currently very active and several experimental prototypes and commercial systems have been implemented. However there are several research problems in this area yet to be investigated, like query optimization , dynamic method definition, invocation, and optimization of method execution. Some of the important prototypes and commercial systems are as follows:

The OODBMS products are GemStone, IDB Object Database, Itasca, Matisse, O2, Objectivity/DB, ObjectStore, Ontos DB, POET, and Versant. The object oriented languages are Eiffel and C++.

1.1 Object Oriented World

A software system constitutes a model of the real world and contains specific information needed in an application[1,2]. Hence, the process of software development can be viewed as the process of building an accurate model of the real-world situation at hand[3]. Since the complexity of software is increasing, a lot of research activities took place on the development of models that naturally and directly reflect the user's conception of the real world at hand[4]. Such a model which ease the task of modeller as-well-as the user system interaction, accelerates the process of software development as a whole. One of the many research directions that have emerged in the last few years with the purpose of achieving this goal circles around the concept of object-orientation.

1.2 User Point of View of Object-Orientation

The aim in technological improvements is to ease the life of ordinary people. A TV-set owner does not bother with the technology inside the TV-set; he/she just wants to use it easily. Object orientation will be explained through a TV-set example,

A TV-set has some common characteristics from object orientation point of view.

- Identity,
- Attributes ,
- Methods.

Identity of a TV-set is trademark, product model etc., along with being a TV-set object; which is an instance of a large class of TVs.

Attributes of a TV-set are color, audio volume, channel frequency etc., whose values or states can be changed by methods of TV-set object.

Methods are the producer supported user interface to change the values or states of attributes of TV-set object, which are implemented to be as simple as possible.

With this abstraction, a TV-set is an object of TV class which has some basic attributes known by everybody and these attributes can be changed with use of user interface methods which are implemented to be as simple as possible.

Therefore a user is supposed to know the functionality and common attributes of a class of objects (i.e., TV-set, car, radio etc.) in order to be able to use them according to his/her needs.

Computer systems with their increasing ability on user interaction (i.e., multi-media) are volunteers to be one of the above large class of objects in use by ordinary man.

1.3 Requirements of Object Oriented DBMS Kernel

Conventional Database systems like ORACLE or INFORMIX are being used for textual information storage and retrieval for a long time. Systems that are implemented for storing data textually can not meet the requirements of much of the organizations today. Since the hardware technology enabled computers to store, transfer and process huge amounts of data, the data types used by organizations are changing.

The power of object oriented database systems lie in their capability to store, retrieve and operate on dynamically defined data types. Whereas conventional database systems have some limited basic data types and far from meeting today's requirements.

An example will be helpful to explain the main difference between object oriented database systems and conventional ones. User of an object oriented database system can define a new class called MAP to the system and also define some operations (methods) on this class, such as locating a point on the map. MAP class may have some textual information along with the image of a map. The user now is able to store different maps in the system and perform the queries on those maps by using his/her methods defined for the class MAP. Whereas this is not possible on conventional database systems , which have some predefined functions for their basic data types, and those functions are not extensible.

An Object Oriented Database (OODB) Kernel should support functionality's such as dynamic definition of new data types (i.e., classes) and methods (i.e., functions) for the manipulation of data types described above.

1.4 Outline of The Study

This thesis is organized as follows:

An overview of MOOD Kernel will be given in Chapter II. An overview of the object-oriented concepts will be presented in Chapter III. Chapter III also gives a comparative analysis of the implementation of the object-oriented concepts in current systems. The design and implementation of the catalog management in MOOD is described in Chapter IV. Dynamic Function Linker (DFL) design and implementation, which makes MOOD Kernel a dynamically reconfigurable system, will be discussed in Chapter V. Conclusion will be given in Chapter VI.

CHAPTER II

OVERVIEW OF MOOD THE KERNEL

The commercial relational DBMSs like INFORMIX and ORACLE provide kernels (INFORMIX On-line engine[5] and ORACLE kernel[6]) that support the following functions:

- management of storage and definition of data,
- controlling and limiting data access and concurrency,
- allowing backup and recovery of data,
- interpreting SQL statements.

Above commercial systems support catalog management for handling data representations at run-time.

The main problem in designing a kernel for an object-oriented DBMS is the late binding of methods or dynamic configuration change. Since database environment enforces run-time modification of schema and objects, late binding is essential. Some of the possible solutions to this problem are as follows:

The first alternative is building a system which uses a persistent language (i.e., C*[7]) as a base language. All other interfaces generate persistent language source as their output. These sources are compiled externally. The compiled programs may be executed separately, or may be activated by using a dynamic linker(dld).

The disadvantage of this alternative is that it is completely contradictory to the object-oriented paradigm and the nature of the database system. The advantage is to be able to use the full power of a programming language (i.e., C++).

The second alternative is to use a full language interpreter (e.g., a C++ interpreter) and extend it with DBMS functionality. The main problem of this alternative is the performance decrease of the system due to interpretation. The advantage is again to be able to use the full power of programming language (C++).

A third alternative is proposed in this thesis, where there is a division of labor between a SQL interpreter and C++ compiler. In this approach there is a uniform SQL-based interface in accessing the database, and the SQL statements are interpreted by the Kernel. But the member functions of the classes (or types) are not interpreted. They are compiled with C++ and used by SQL interpreter through a dynamic function linker. The advantage of this approach is that the interpretation of the functions are avoided increasing the efficiency of the system.

A database kernel is implemented by using above approach as a part of Metu Object Oriented DBMS (MOOD) project. MOOD also has a SQL like object-oriented query language. MOOD data types are derived from C++ data types and therefore there is no impedance mismatch.

In this implementation a tool kit system EXODUS [8] storage manager (ESM) is used in which following functionalities are readily provided;

- management of storage,
- controlling and limiting data access and concurrency,
- allowing backup and recovery of data.

Additional functions implemented by the MOOD kernel are

- interpreting SQL statements ,
- definition of data.

SQL interpreter , handling of the data definition (catalog management) and dynamic function linking will be described in next three sections.

2.1 SQL interpreter (MOODSQL)

MOODSQL is responsible from :

- optimization of MOODSQL queries,
- Interpretation of arithmetic and boolean expressions,
- Dynamic definition and linking of member functions.

MOODSQL interpreter cooperates with the MOOD catalog manager and the MOOD dynamic function linker. SQL syntax is enriched to support MOOD data model. As an example, MOODSQL supports path expressions which are not supported by the standard SQL:

```
MOODSQL> select e.dept.floor
```

```
>              from employee e
```

where the employee class contains a reference to department class trough dept attribute.

As well as supporting simple type of queries, MOODSQL supports composite expression evaluation with dynamic function linking. As an example :

```
MOODSQL> select e.ename
>              from employee e
>              where e.raise_salary(0.3) < 2000000 and
>              e.dept = "CC" ;
```

where raise_salary is a method of employee class which raises the salary of employee.

2.2 An Overview of Handling Data Definition

In MOOD, data can be defined through MOOD data definition language or through C++. When data is defined through MOOD data definition language, the definitions are stored in the Catalog and a C++ header file is created for future use by dynamic function linker. C++ preprocessor (Cfront) is modified such that it extracts the catalog information and stores into the Catalog.

The basic types supported by the MOOD are Char, Boolean, Integer, Float, Double and String. The type constructors are Tuple, Set, List, and Ref. Any complex type may be created by using basic types and recursive application of the type constructors.

Any type (or Class) in the system has a unique type identifier and a name. The functions typeId(char *typeName) and typeName(int typeId) return type identifier and name of the type (or class) respectively.

Differences between a type and a class from the implementation point of view are:

- Classes have a default extent(storage) that contains the related instances
- Instances of the classes can be shared through their object identifiers,
- Values which are instances of types have copy semantic,
- Classes are organized into a class lattice.

The catalog contains the definition of classes, types, and member functions in a structure similar to a compiler symbol table. In order to achieve late binding at run time, it is necessary to carry compile time information to run time. This is accomplished by the use of the classes MoodsType, MoodsAttribute and MoodsFunction and their supported methods. The MoodsType class keeps track of all the types used in the system. The MoodsAttribute stores the information about attributes of classes. The MoodsFunction class keep information about member functions. All of these classes are called system classes. Implementation of these classes are described in Chapter V.

2.3 An Overview of Dynamic Function Definition and Linking

The power of object oriented applications lie in allowing the dynamic schema changes. This can be achieved by interpreting the code as it is done in Smalltalk[9], where a user can add classes at any time by using existing classes' characteristics. The created class, is a part of the whole system (i.e.,schema of system changes dynamically).

Another way to change system schema is to add a class and recompile the whole system to link the class's member functions to the new system. This, however causes the system to be unavailable for a time period .

With the approach in this thesis a user can define classes and member functions through SQL interpreter. Methods of classes are defined in C++ language. A method obtains class definition from the class header file automatically. This implementation leads to dynamic schema changes without recompilation or interpretation. The only cost is the preprocessing and compilation of the defined functions. An example of dynamic function definition and linking is given below.

Dynamic function definition:

```
MOODSQL> create function "function-C++-source-file" ;
```

since function source has class name in its definition (C++ syntax), this information is extracted and stored into the catalog while making some preprocessing on the function.

Thus a member function defined for a class can be called through an SQL query immediately after its definition.

CHAPTER III

AN OVERVIEW AND CLASSIFICATION OF OBJECT ORIENTED SYSTEMS

In[10], the object-oriented concepts has been defined and the existing object-oriented systems has been classified according to how each system has implemented these concepts. This work created an excellent resource to compare the object-oriented concepts implemented in the MOOD system with other implementations and to put MOOD system into a perspective.

3.1 Objects and their Properties

The Entity-Relationship data model, modelled the universe of discourse to consist of entities and relationships. An entity has the following characteristics:

- it is identifiable,
- it is relevant (of interest),
- it is describable, i.e. have characteristics.

The prime characteristics of entities refer to their state (or value). Entities may also have some meaningful relationships with other entities for organizational point of view. Furthermore, activities (i.e. operations on entities) occur over time, resulting changes to the entities' states and to their interrelationships. Both the entity states and the activities are subject to constraints which more precisely define the characteristics, thereby distinguishing 'reality' from other possible worlds[11].

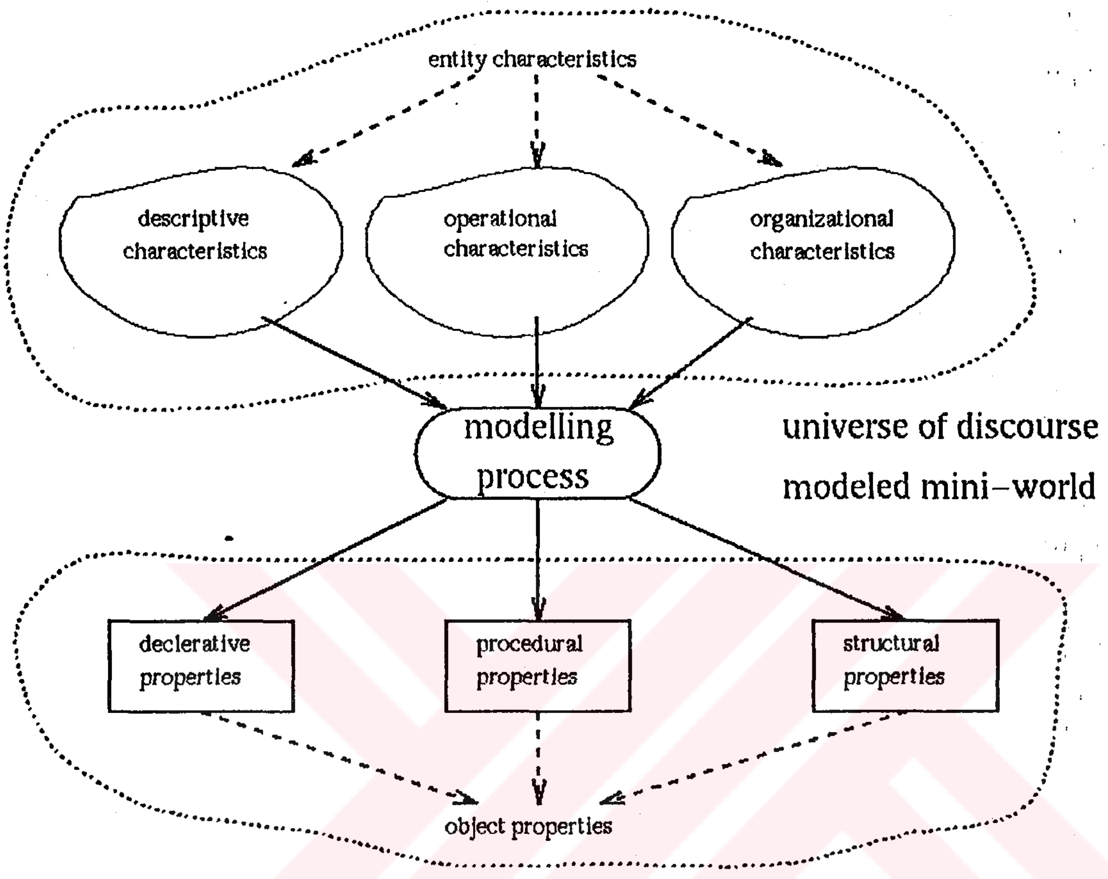


Figure 3.1 State-oriented view of modeling

The state oriented view of the world given in Figure 3.1[10], shows the conceptual primitives, i.e., entities, having descriptive, operational, and organizational characteristics as mentioned above. By means of the modeling process, each entity relevant to the current universe of discourse is represented by a corresponding object in the model. Thus, when talking in terms of the mini-world model, the notion of object is used as the symbolic representation of a real world entity.

Objects have the following properties:

- Declarative properties are used for descriptive purposes, e.g. attributes.
- Procedural properties are applied for operational purposes,e.g.methods.
- Structural properties are employed for organizational organizational purposes. They represent relationships among entities e.g., abstractions

3.2 Object Specification

A tree-like classification scheme is used to express the building rules of the object specifications as shown in Figure 3.2[10]. Each criterion of classification is given on the left side of the Figure. Sometimes it is not possible to classify a system uniquely because it has different versions , hence classifications made from this point on may have different systems shown at the leaf level.

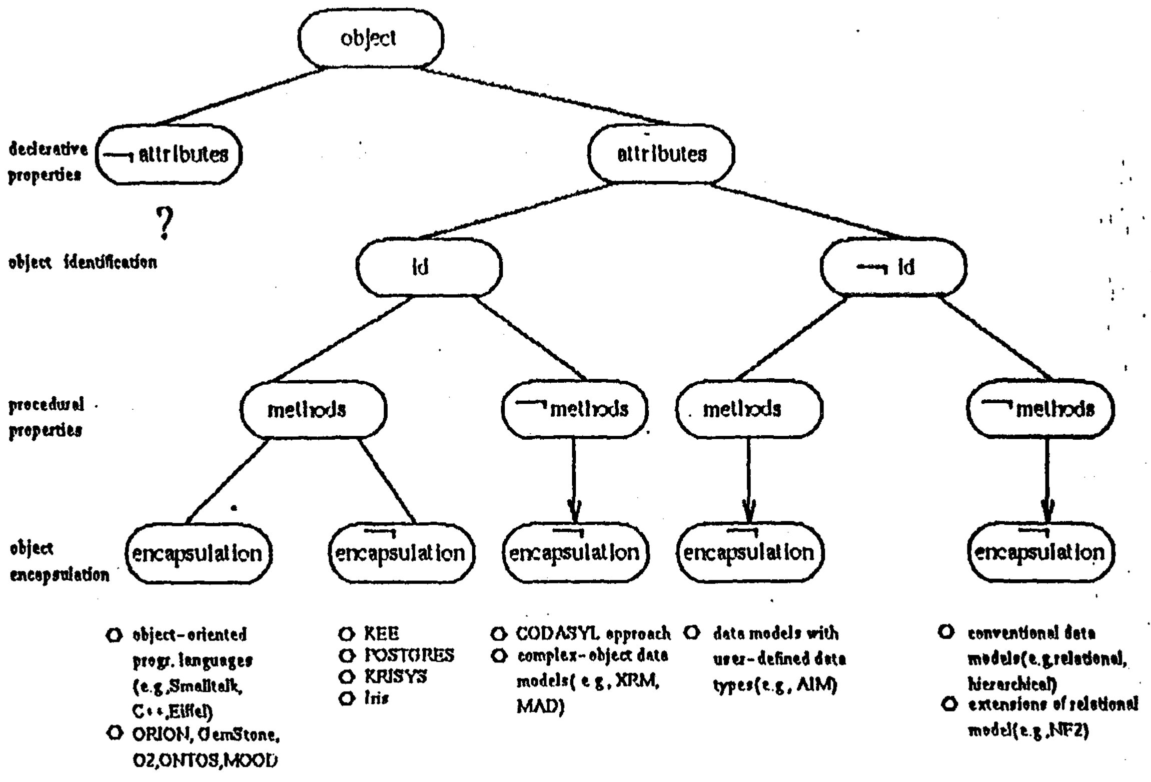

object
declarative properties
¬ attributes
attributes
?
object identification
id
¬ id
procedural properties
methods
¬ methods
methods
¬ methods
object encapsulation
encapsulation
¬ encapsulation
¬ encapsulation
¬ encapsulation
¬ encapsulation
O object-oriented progr. languages (e.g. Smalltalk, C++, Eiffel)
O ORION, GemStone, O2, ONTOS, MOOD
O KEE
O POSTGRES
O KRISYS
O Iris
O CODASYL approach
O complex-object data models (e.g. XRM, MAD)
O data models with user-defined data types (e.g. AIM)
O conventional data models (e.g. relational, hierarchical)
O extensions of relational model (e.g. NF2)

Figure 3.2 Concept terrain for objects

At the first level of the tree, declarative properties of the systems are compared. Since there is no system, which does not support attribute definitions, known up to now, and it is shown with a question mark on Figure 3.2.

Object identification level of the tree shows whether object identity is supported or not [12]. Object identification implies that an object has a valid object identifier(OID) as long as the object exists and that it is neither updatetable or reusable. Apart from CODASYL approach (and its database key concept[13]) conventional data models [14] and straightforward extensions (e.g., NF^2 [15,16]) do not have OID concept.

Procedural properties introduce the alternative of having methods attached to objects or not. Methods are either system or user defined procedures that provide access to the attributes of a either user or system defined defined object, i.e., to all information within an object.

Object encapsulation means there is no other way to access or change object attributes except through methods defined on the object, i.e., if there is no method defined for updating an attribute of the object , that attribute can not be updated. Here is an example from C++ :

```
class TV {
private:
        unsigned volume;
        unsigned frequency;
public:
        set_volume( unsigned Pvolume ) {
                volume =  Pvolume ;
        } ;
        set_frequency( unsigned Pfrequency ) {
                frequency = Pfrequency;
        };
};
```

This is an encapsulated class , in which attributes volume and frequency can be changed only by set_volume and set_frequency methods defined for the class. Note that attributes of the class are defined as private and therefore it is not possible to make assignments to these attributes. However member functions are defined as public and can thus be called any time after creating an instance of class TV.

```
main () {

.

TV my_tv(); // An instance of class created.
my_tv.volume = 100 ; // this is not allowed.


};
```

If the object concept does support methods, but not object identification, it is not possible to implement encapsulation since there would be no way to address an object, in other words, if the object can not be located , it is meaningless to support methods for that object, since system can not locate specific object to apply method. Therefore encapsulation is not possible in this case.

Encapsulation concerns both attributes and methods, but in a different way. Also, encapsulation can be total or partial. In the case of methods, systems may have only public methods or may allow to declare some of them private; total encapsulation does not make sense. Attributes are either all hidden or can be made visible selectively; the case where all attributes are explicitly accessible implies "no encapsulation".The fact that whether attributes are hidden by default and can be made visible(through an "export" or "public" declaration, as in Eiffel[17], ONTOS[18], and C++[19], respectively), or whether they are accessible by default and can be hidden (AIM[20]); is not distinguished. If there are visible attributes, types of access can also be read only or update. While a C++ public attribute can not be write-protected, Eiffel exported attributes cannot be modified. Only O2[21] offers both of the alternatives.

3.4 Attributes: declarative properties

Classifications of the object-oriented systems according to attribute definition is given in Figure 3.3.

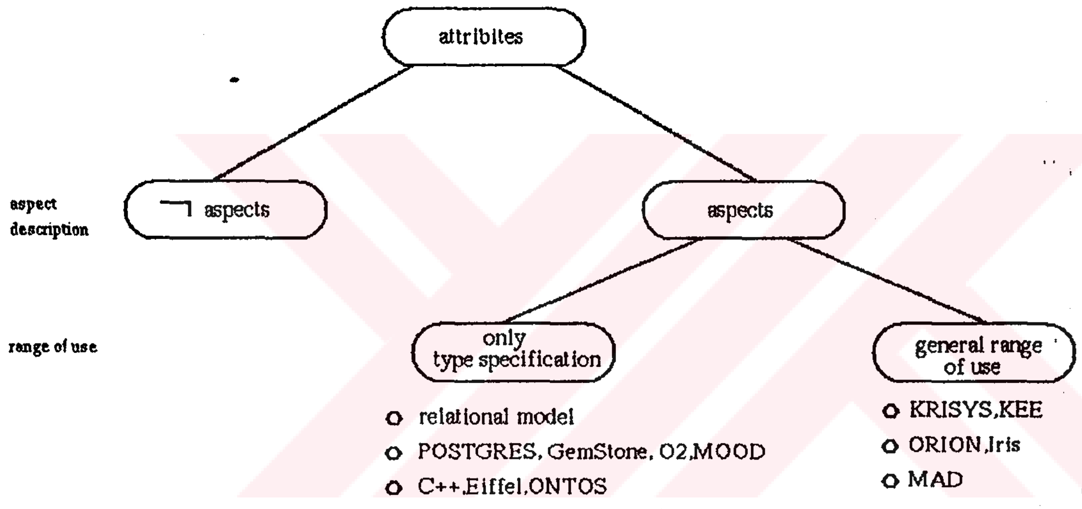


Figure 3.3 Attribute specification

Aspects are defined as facilities to specify some constraints on declarative properties such as possible values, default, cardinality, etc. They can be used only to specify an attribute's type as it is in the case of systems with typed attributes (e.g., the relational model, C++, ONTOS, POSTGRES[22], GemStone and MOOD). Here, it is not distinguished whether the type specification is optional , as in the case of

GemStone, or whether it is mandatory as in the case of MOOD or C++. Through aspects it is also possible to explicitly define integrity constraints. For example, for cardinality restrictions, default values, user-defined specifications, can be defined as in the case of ORION[23].

3.5 Methods: procedural properties

Methods are defined for protection (encapsulation) of the declarative properties of objects from being read and manipulated in an arbitrary way. Furthermore, methods provide more abstract and problem oriented ways of working with objects.

If methods are used for accessing object properties, this guarantees consistent state transitions supposing that methods work correctly. Objects hide the number of internal attributes and storage representations from outside world in this manner.

A method can also send messages to other methods, which is a very powerfull mechanism. But if misused, there is a disadvantage in calling a method within a method. Since, state changes may propagate through network of objects in an uncontrollable manner. Therefore the user, who writes method code should be very careful in considering the side-effects.

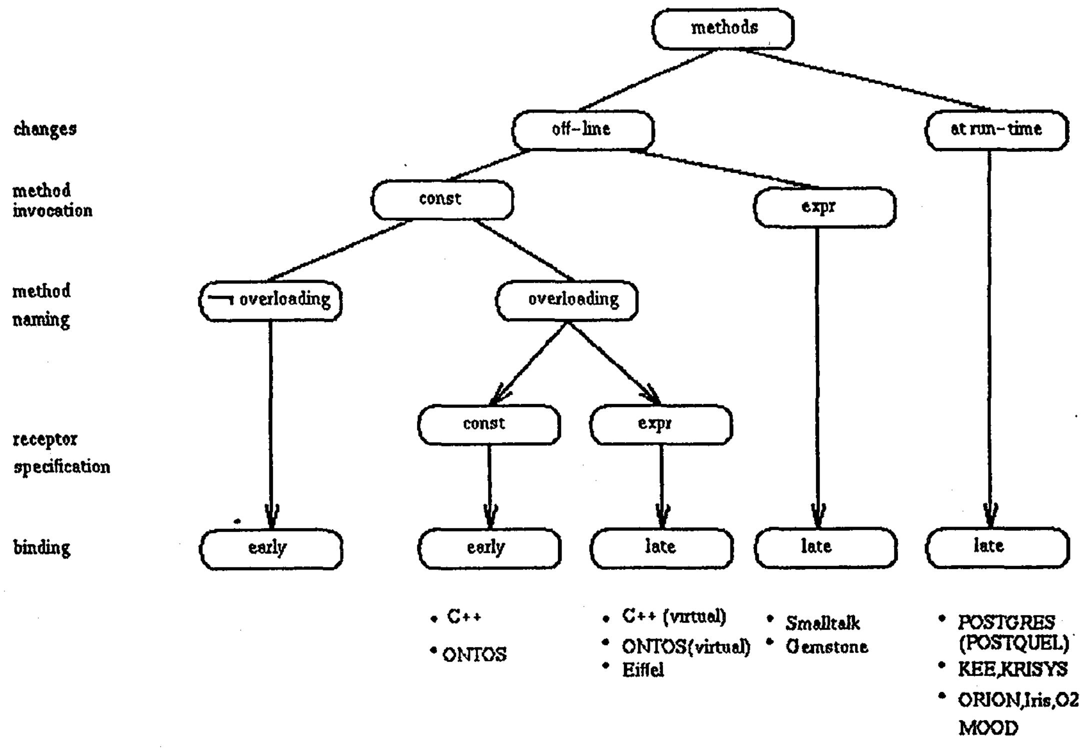


Figure 3.4 Concept terrain for Methods and Binding

As depicted in Figure 3.4[10], changes on methods can be made either at run-time or offline. Run-time changes mean that, a method code can be changed, inserted and deleted by the run-time system. Whereas, with offline changes, operations on methods described above can only be performed in some other state of the system (i.e., development state, uncompiled state etc.).

Method identification mechanisms of the systems may also differ in the naming or overloading of method names. Some systems restricts name of methods to be unique, whereas others may accept methods with the same name to the system. C++ has the facility of overloading, i.e. two methods of a class can take same name, with different number or type of parameters. Systems having overloading facility use some techniques to name methods. As an example C++ uses the signature concept, which changes name of the method by appending class name (on which the method is defined) , parameter types and return type to the signature of the method and identifies method with this signature throughout the system.

One of the important aspect of method invocation is binding, which describes the process of locating the procedure code of an object method, given a class and a method name. The binding can be achieved at compile time('early-binding') or run-time('late-binding').

If modification and binding of methods is handled at run-time(late-binding), implementation should support expression type of parameters to be passed to the methods.

3.6 Object Abstractions: structural properties

Abstraction means the representation of relations between objects. Object oriented data models propose terms such as class, set, aggregate etc. to construct relationships between objects.

Real world objects can be classified according to their properties. The basic unit of structural properties, that is, classes will be described in the next section. The set and aggregate structures of the object oriented paradigm will be discussed in sections 3.7 and 3.8.

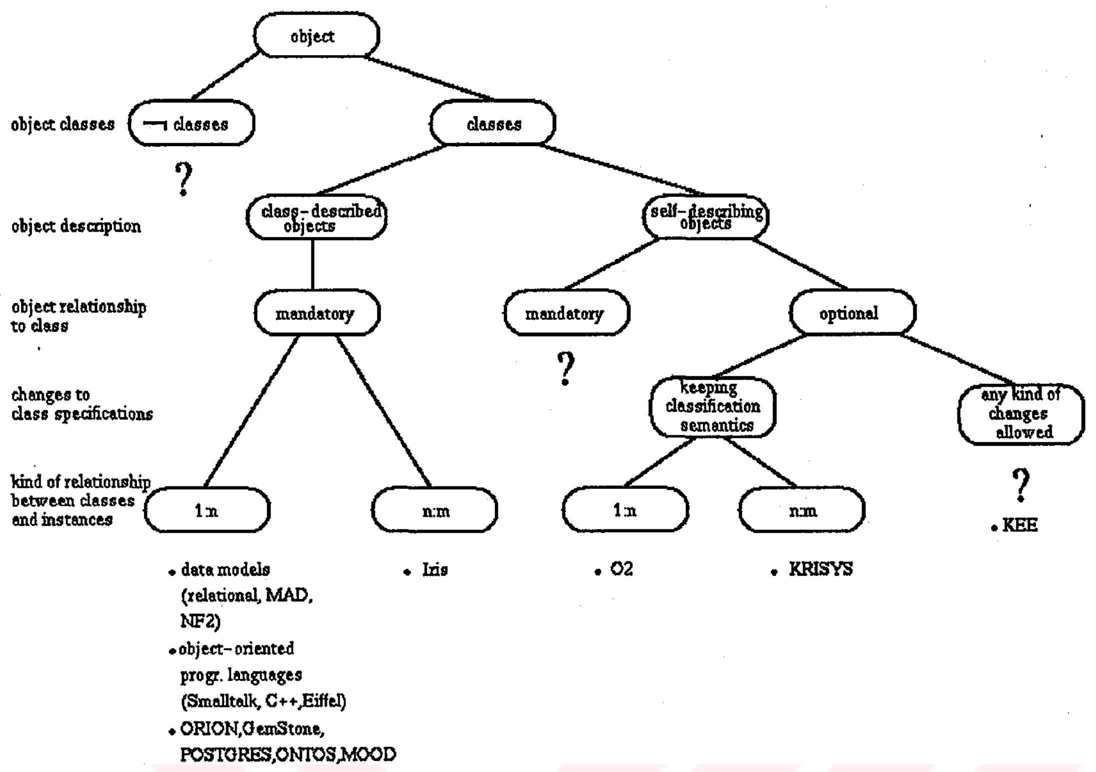
3.6.1 Classes

Classification is the simplest grouping method for object specification. With this abstraction, classes provide generic information in the model by allowing to refer objects are the members of the same class as a prototype or as a representative. For example, EMPLOYEE class may be introduced to the system which holds common information about all employees(instances) of a company.i.e., each employee of the company has same instance variables and methods such as age, salary, position etc.

3.6.1.1 Handling Classes in a Model

Object-oriented designs consider classes in two different ways. The first category of systems limit the class definition as the type specification for objects derived from the class, i.e., all attributes and constraints defined by the class must be pertinent and applicable to all instances of the class. With this approach, the object relationship to classes is mandatory as shown in Figure 3.5. In the second approach, classes, like any other defined object in the system, are ordinary objects. With this approach, relationship of objects to classes can be optional as well as mandatory.

If the behavior of instances is determined by classes, all instances must act in the same way, consequently modifications of methods must be prohibited since methods are related with all instances. But if classes have only the role of specification of the interface of methods, changes on methods do not conflict with this semantic of classification and may be allowed.



object
object classes
¬ classes
classes
?
object description
class-described objects
self-describing objects
object relationship to class
mandatory
mandatory
optional
?
changes to class specifications
keeping classification semantics
any kind of changes allowed
kind of relationship between classes and instances
1:n
n:m
1:n
n:m
?
• KEE
• data models (relational, MAD, NF2)
• object-oriented progr. languages (Smalltalk, C++,Eiffel)
• ORION,GemStone, POSTGRES,ONTOS,MOOD
• Iris
• O2
• KRISYS

Figure 3.5 Object Classes and their relationship to instances

If relationship of object to a class is optional, such systems are classified according to their object definition's degree of freedom from class definitions. Some of the systems permit objects change class definitions by keeping class semantics such as O2 and KRISYS[24] , but others let objects make any kind of change on class specifications as in KEE.

Most of the DB systems use classes as meta-data or type definitions and objects belonging to those classes as instances. With this approach class definitions or meta-data refer to DB-schema and instances or data refer the DB data. The relation between classes and instances in the above semantic is 1:n.

If an instance holds characteristics of more than one class, the relationship between classes and instances is said to be n:m, systems like Iris[25] and KRISYS have this characteristic as shown in Figure 3.5.

3.6.2 Class Hierarchies and Inheritance

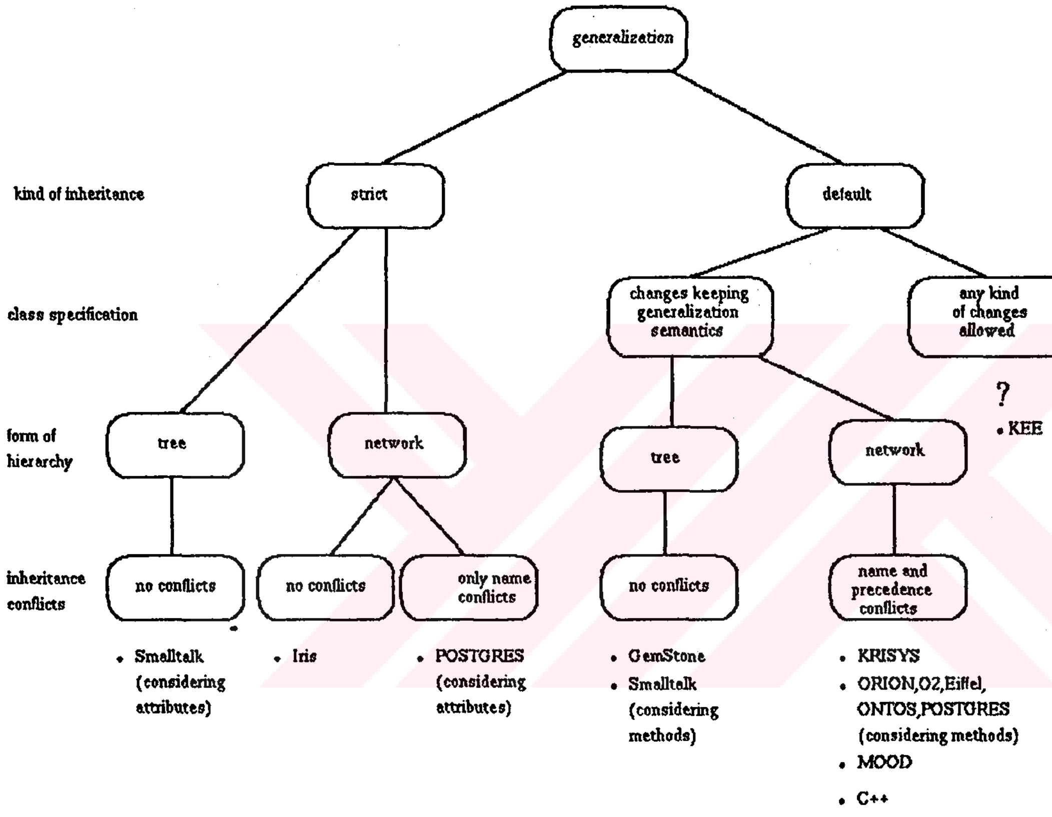

generalization
kind of inheritance
strict
default
class specification
changes keeping generalization semantics
any kind of changes allowed
?
• KEE
form of hierarchy
tree
network
tree
network
inheritance conflicts
no conflicts
no conflicts
only name conflicts
no conflicts
name and precedence conflicts
• Smalltalk (considering attributes)
• Iris
• POSTGRES (considering attributes)
• GemStone
• Smalltalk (considering methods)
• KRISYS
• ORION,O2,Eiffel, ONTOS,POSTGRES (considering methods)
• MOOD
• C++

Figure 3.6 Class hierarchies

In modelling objects with classification, modeller needs to represent structural relationships between classes. One of the most common ways of doing this is, to form relations between classes of objects such as , sub-class of, subcomponentof or hierarchy relations. In this section hierarchy relations between objects will be described with inheritance concept.

In the class hierarchies, some of the super classes's characteristics are carried to the sub classes, this is called inheritance. Inheritance mechanisms may change from system to system. Inheritance can be default as well as strict which will be described in the next section. Also classes may have multiple or single inheritance mechanisms in the class hierarchy, which will be discussed in section ??.

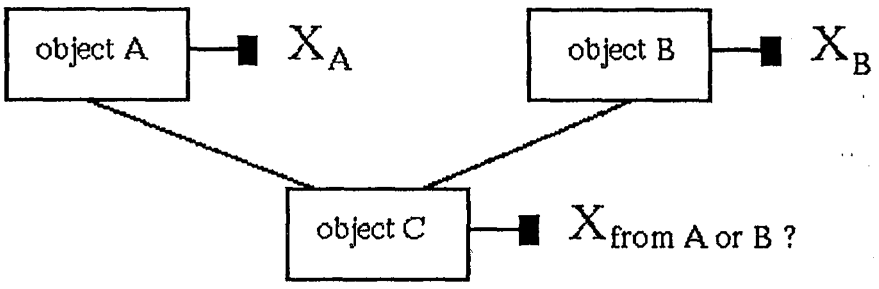
3.6.2.1 Default vs. Strict Inheritance

Inheritance of classes are separated in to two parts according to heritages sub-classes receive from the super-classes. If a sub-class can modify the inherited properties of the super-classes, such as attribute and method definitions, this type of inheritance is called default inheritance. On the other hand, if any sub-class can not change the properties it received from the super-classes, this type of inheritance is called strict inheritance.

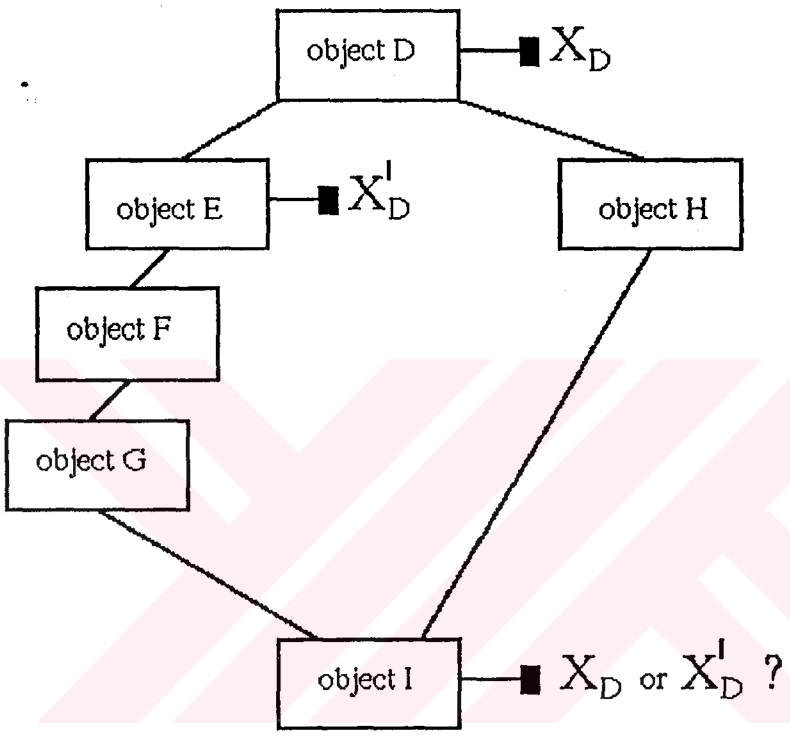
In most of the systems, methods and attributes are inherited with default inheritance mechanism, hence a sub-class may re-define any methods defined by its super-classes according to its properties. Whereas , strict inheritance does not let sub-class re-define a method or attribute defined by one of its super-classes.

3.6.2.2 Single vs. Multiple Inheritance

There are two forms of class hierarchies. Actually, the form of hierarchy determines the amount of heritage received by an object. For example, systems providing tree-like hierarchies allow for single-inheritance, i.e., objects receiving characteristics from only one class. Whereas the other systems which have a lattice type(network) hierarchies, allow multiple inheritance, i.e., objects receiving characteristics from several classes (e.g., POSTGRES, ORION, O2, ONTOS, KRISYS, KEE, MOOD).



a) example of name conflict



b) example of precedence conflicts

Figure 3.7 Multiple-inheritance Conflicts

Multiple inheritance characteristic of the model imposes name and precedence conflicts in inheritance schema, which is depicted in Figure 3.7. Name conflicts occur when two attributes, methods, or aspects with the same name have been defined for distinct classes of a network hierarchy and are inherited by a common class This occurs in the systems which does not allow two attributes or methods of a class to be same. Precedence conflicts can only occur in default inheritance hierarchies, if an attribute or method is once defined by a class in the hierarchy and updated by its subclass, system should choose one of the characteristics as depicted in Figure 3.7b.

3.7 Sets

Grouping of objects may not only necessary because they have the same properties only, but sometimes it is necessary to describe the properties of the group of objects as a whole. Set association between objects is used in such a case. Set concept originates from the set theory, so that it does not allow existence of duplicate elements.

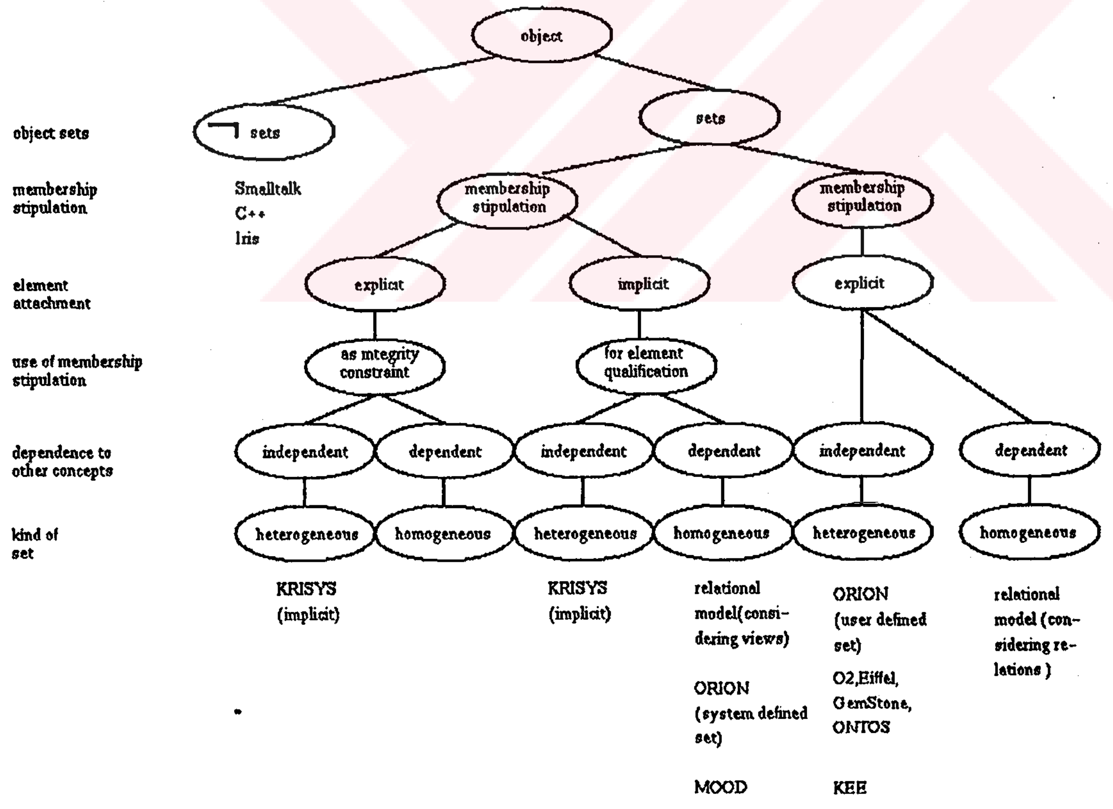


Figure 3.8 Sets

3.7.1 Handling Sets in a Model

If sets are to be supported by an object-oriented system, the set construction mechanisms should be provided. The set characteristics may be defined by system implicitly as well as explicitly by the user.

There are two ways of member attachments to a set. Member qualification or attachment to a set may be handled by the system according to a given criterion on the construction of the set object or attachment can be done explicitly by the user through a query or selection mechanism.

Set representation of the object-oriented model may be bound to a class abstraction, in this case, as shown in Figure 3.8, there is a dependence to a class definition, hence the set can only contain homogeneous objects.

3.7.2 Set Hierarchies

Some of the object-oriented system designs consider set hierarchies in their data model, which means , the user can create a new set of objects in the system by narrowing the criterion of the previously defined sets. Hence a set hierarchy is constructed.

3.8 Aggregates

Frequently, there is a need to treat objects not as atomic entities, as classification, generalization and association do, but as a composition of other objects. Therefore, in modelling real-world entities, a way to represent part-of relations in the model is needed. The aggregation concept of object-oriented paradigm supports modelling of part-of relations between objects.

Systems are classified according to their support of aggregation concept in the first level of the classification tree shown in Figure 3.9[10]. In fact, since aggregate relations are used to represent part-of hierarchies between real-world entities, there should be some kind of hierarchy in representing aggregate relations between modelled objects of the system. Hence, there is not any known system which supports aggregate construction but not aggregate hierarchies.

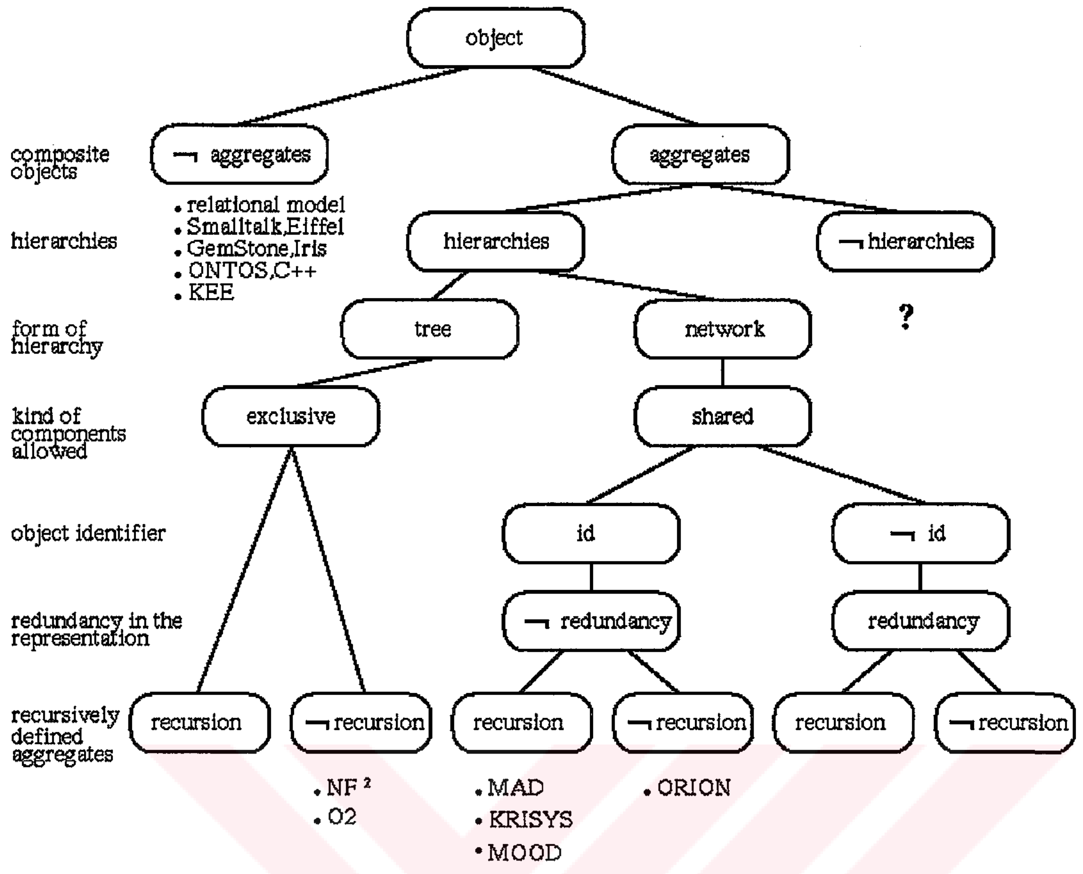


Figure 3.9 Aggregation

Aggregate hierarchies supported by the systems in tree or lattice(network) structures are as shown in Figure 3.9[10]. In systems, those support tree like hierarchies an object can take part in one aggregate construction, (labeled as exclusive) whereas if a system supports lattice or network like hierarchies, objects can be used in constructing of any number of aggregate objects(labeled as shared in Figure 3.9)

If object identity is not supported by the implementation of the system, objects' definitions, used in the aggregation should be copied to the aggregate object which causes redundancy in definition of aggregate object as shown in Figure 3.9.

Whereas if object identity is supported by the implementation, only object identifiers are refered from aggregate object without copying characteristics of component objects.

If an object is formed by aggregation, it can also be used to form other aggregate objects, this capability of the model is termed as recursive aggregation. Real-world entities can be described naturally with recursive aggregate definitions. For example, a TV-set has some electronic and electrical components which are aggregately form TV-set object, and electronic components are also formed by some other components again with aggregation.

In recursively aggregated objects, the user just sees that object and if he is interested in, he can obtain detailed information on the components of the object. This is the logical formation of the aggregate objects.

3.8.1 Dependent vs. Independent Components

The part-of relations may be classified into more restrictive forms or constraints in the design such as dependent or independent.

In the case of dependent part-of relations, if a part is deleted from the system, all parts, which are used as components of the deleted part are deleted by the system automatically. i.e., if TV-set object is deleted from the system, all electronic and electrical parts and their components are deleted from the system. Whereas, if part-of relation or aggregation is independent, only the requested object is deleted from the system and the objects which use deleted object in their construction are not deleted along with the components, which form the deleted object.

3.8 Handling Different Object Roles

Objects in an object-oriented system may take more than one role in their lifetime, such as they may be instances of classes, elements of sets or components of aggregations. The object roles are classified in Figure 3.10 .

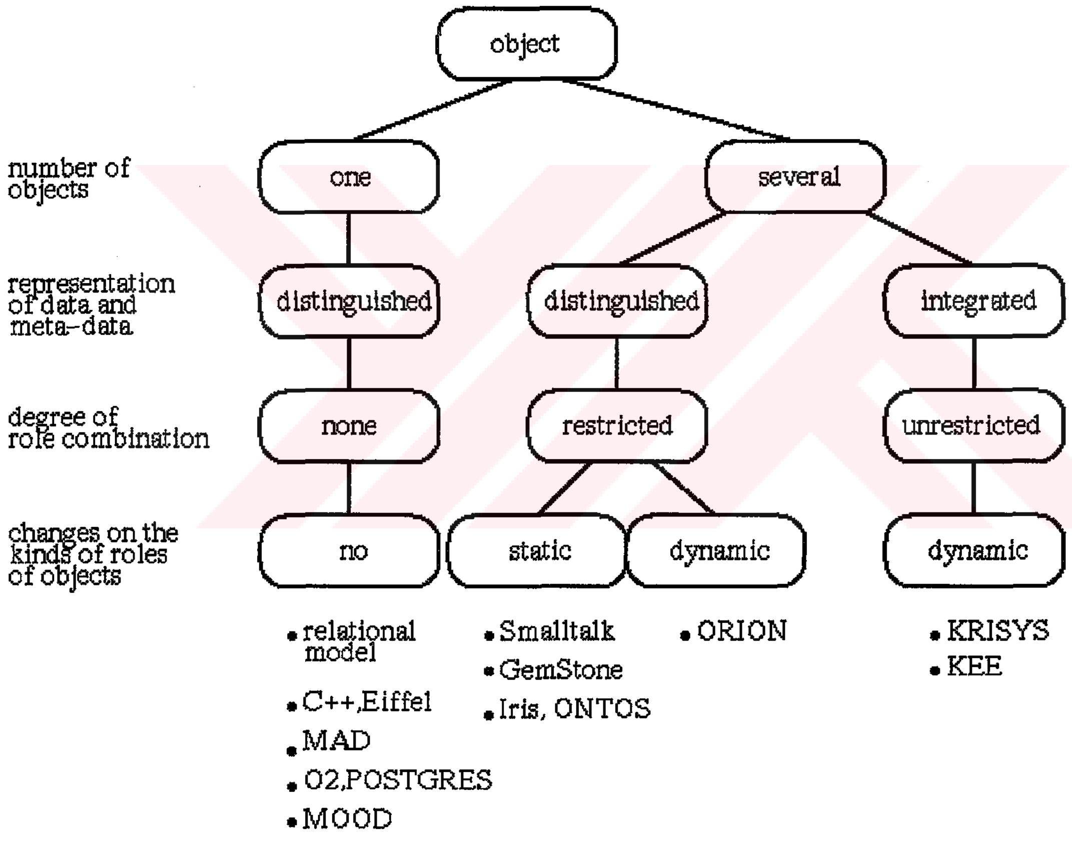


Figure 3.10 Integration of abstraction roles

In the first level of the tree shown in Figure 3.10, systems are classified in their capabilities to give objects one or several roles.

Systems that are giving objects just one role must distinguish meta-data and data representations, there is no role combinations that can be established on a single object, i.e., object roles are defined before they are created and not changed during their life time. On the other hand, if a system can support several role combinations on an object, it may distinguish the representation of meta-data and data.

In the latter approach, in which objects may have different roles, next level of classification is whether the representation of meta-data and data is distinguished or integrated. In the first case, degree of role combinations that can be applied to an object is restricted by meta-data definitions, as in the case of ORION system, instances may be either components of other objects or aggregates, but not classes. Whereas , if the representation of meta-data and data is integrated, all objects in the system are treated in the same way and object semantics are kept within objects, i.e., 'self-describing' objects.

On the next level of the tree shown in Figure 3.10[10], the changes on the kinds of roles of objects are considered and classified. Static changes mean that, the role combinations are given to the object at creation time, and can not be changed thereafter. Whereas if dynamic changes are allowed by the system, object roles may change at any time after creation of the object.

3.9. Query Languages

One of the main properties of object-oriented systems is encapsulation, hence using the methods of the object to access its internal structure. In this sense queries on object attributes should be carried out by the methods of the class. Design rationale makes it difficult for the object-oriented database management systems to support encapsulation in their query languages, since it is not realistic to make a query execute a method on every instance to receive an attribute value, or join attributes of instances. But encapsulation is supported when the database is accessed through the programming language. Above approach is used in most of the object-oriented database management systems like O2 and MOOD.

CHAPTER IV

MOOD KERNEL DESIGN AND MOOD CATALOG MANAGER

4.1 Introduction to MOOD Kernel Design and MOOD Catalog Manager

Design of the MOOD Kernel and MOOD Catalog Manager (CM) is discussed in this Chapter. The implementation of object-oriented concepts discussed in Chapter III will be explained.

The first section describes the Exodus Storage Manager(ESM) architecture, which is used as storage manager of the system, and supports functionalities described in section 2.1 of Chapter II.

The CM design supports some of the functionalities presented in Chapter III, with the proposed data model of MOOD. Each aspect supported by the CM will be discussed in the section 4.3, implemented kernel will be discussed in the section 4.4 and the section 4.5 is devoted to the implementation aspects that are left as future work.

4.2 Exodus Storage Manager (ESM)

In this section ESM architectural overview is described briefly.

ESM has a client-server architecture. Application programs call routines in the 'client module' of the storage manager to access and manipulate data in storage objects. The client module of the storage manager cooperates with a server process to access and manipulate objects and the files that contain them. Currently, clients can only connect to one server at a time; i.e., distributed transactions are not supported. The ESM server is a separate and possibly remote process which provides a variety of services to multiple clients. These services include I/O, transactions, concurrency control, recovery, indexes, and files.

4.2.1 Client Module

The ESM client module is a library which is linked with an application program. The interface for this library provides routines for creating, destroying, reading, writing, inserting into, deleting from, and versioning of objects. Routines are also provided for creating, destroying, accessing and modifying files of objects and indexes. Initialization, administration, and transaction support routines are included as well. It is important to distinguish between the application program and the client module. The application program only calls the interface routines of the client module. It does not access the server directly. The client module supports the interface routines and communicates with the server as needed. The term client will be used in referring the ESM client module hereafter.

4.2.1.1 Architecture

The client module has its own buffer pool, which is used in chaching pages containing objects and indexes. Object and index operations are performed by the client module in its buffer pool. This contrasts with other client-server architectures where the client simply sends a request to the server to change the object instead of performing the operation itself. File operations are performed on the server. For transaction management, the client has a lock cache, some transaction information, and a logging subsystem. The client communicates with the server's transaction and lock managers to maintain the lock cache and transaction information. The client's logging subsystem generates log records for the operations that it performs and sends the log records to the server's log manager.

4.2.1.2 Operation

An application begins by performing some initialization and then starts a transaction. During the scope of transaction, various files, objects, and indexes can be accessed and changed. The application can then commit or abort the transaction then start another one. After transaction processing is complete, the application shuts down the client.

A client application process begins by calling sm_Initialize(). This statement initializes data structures and connects the client to the desired server process. Once connected, the client asks the server for the values of certain parameters such as the log page size. Before accessing data, the application must request that relevant disk volume(s) be mounted by the server.

Once initialization is complete the application can start running transactions. The client can be in one of three transaction processing states: not running within a transaction, running within a transaction, and an aborted transaction. After initialization, the client is in the first transaction state. When the application begins a transaction, the client module requests a transaction ID (TID) from the server. All future data and lock requests sent to the server in the scope of the transaction will contain this TID.

After an application has started a transaction, it can begin accessing objects, files, and indexes. When an application requests an object that is not found in the client's buffer pool, the client requests the page(s) containing the object from the server. The request also contains the desired lock mode for the object. When application wishes-to change an object, a request for an exclusive lock on the object will be sent to the server.

To support transaction abort and recovery, operations performed on objects and indexes are logged by the client. The ESM uses an adaptation of the ARIES recovery algorithm[26] to support transaction abort and recovery. This algorithm uses a write ahead logging(WAL)[27] protocol.

After performing object, file, and index operations an application will either commit or abort the transaction. Transaction abort may also be initiated by the server, for example, because of a deadlock.

4.2.3 The Server Architecture

ESM server is a multi-threaded process providing I/O, file, transaction, concurrency control, and recovery services to clients. The way the server handles client requests are described in the following.

4.2.3.1 Threads and Request Processing

All request processing within the server is performed within the context of a thread. Server threads are units of execution similar to the co-routines provided by Modula-2. Each thread has its own stack for maintaining its execution state. A thread is always in one of the following states: executing, sitting in an inactive pool, waiting on the ready queue for a chance to continue executing, or waiting on a list for a resource. Resources that a thread may await include locks, latches, semaphores, disk I/O, and a signal from another thread. Thread switching is implemented using the setjmp() and longjmp() functions in the standard C library.

When a request arrives, the server assigns a thread from the inactive pool to handle the request and begins executing the thread. The thread runs until it has to wait for a resource, or voluntarily gives up the CPU, or completes the request, In the mean time it responds to the client and buffer large object pages contiguously. The buffer manager enforces a write-ahead logging protocol. Since the buffer area is shared by multiple threads, latches are obtained for synchronization. Finally, buffer manager provides the concept of 'buffer group' as proposed in the DBMIN buffer management algorithm[28].

The server uses the buffer group facility to allocate buffer space for various purposes. There is one large LRU buffer group used to satisfy client I/O requests. Separate smaller buffer groups are used for reading and writing the log. Smaller buffer groups are also allocated for managing the bitmap pages associated with each disk volume.

4.3 Design of MOOD Kernel

In this section the implementation of the object-oriented concepts defined in Chapter III in MOOD Kernel is presented.

4.3.1 Object Specification In MOOD Kernel

MOOD Kernel supports attributes and object identity. Each object of the system has an Object Identifier (OID) associated with it. This OID is given to the object at creation time by ESM, and it is the disk start address of the object returned by ESM. The OID's of objects are given at creation time, as mentioned before, and not changed thereafter until object is deleted from the system.

User of the MOOD Kernel has the ability to define and use methods(C++ functions) on user classes.

Object encapsulation is divided into two parts, method encapsulation and attribute encapsulation. These encapsulation properties are supported through C++. But in accessing the instances of the database, direct access method is used for the reasons given in[29]. Otherwise for each class, methods in accessing the values as well as updating them would be needed. With the last approach, every class in the system should support methods to insert, delete, get and update attribute value of instances, in which slows down the system. This is because, while making a search on some attribute, for each object in the database, a method should be invoked. This approach is not used for commercial systems like O2.

4.3.2 Declarative properties of MOOD Kernel

Since MOOD Kernel supports attribute concept, attribute specifications (or declarative properties) are compared and discussed in this section.

Aspect concept is defined in section 3.4.1 of Chapter III. MOOD Kernel design limit aspects as type specifications. i.e., each instance in the system should be defined to be a member of a class. In this view, class definitions are analogous to type definitions of a programming language. Other well known database systems, as well as programming languages using this type of aspect definitions are POSTGRES, GemStone[30], O2, C++, Eiffel and ONTOS as depicted in Figure 3.3 in Chapter III.

Class definitions in MOOD Kernel provide a prototype of instances in the database along with the relations with other classes in the system.

4.3.3 Procedural Properties (methods) of MOOD Kernel

Methods can be defined by user to manipulate classes (user-defined classes). In method definition, C++ language is used. Dynamic Function Linker , which is described in next section is used in activating user defined methods.

Method binding is achieved and implemented at run time (late binding). Since C++ supports overloading of method names, overloading is possible with the signature concept defined in section 3.4.2 of Chapter III. Overloading is achieved by adding some information to the method's name such as class name (on which method is defined, i.e., receptor), method name (selector), attribute type information, and return type are concatenated to form the signature of function; this is explained in next Chapter.

4.3.4 Object Abstractions in MOOD Kernel

Instances are grouped in the abstraction level of a Class in other words classes have extensions. Classes in MOOD Kernel have some properties described below.

The relation between classes and instances is a 1:n relation , i.e., under a class there could be any number of instances associated with it, but an instance can not be associated with more than one class. Most of the object oriented systems are implemented with this approach, as depicted in Figure 3.5.

Class inheritance schema of MOOD Kernel is multiple inheritance, i.e., a class may inherit from a number of classes. With this definition of classes, two problem arises; name and precedence conflicts. These problems are solved as explained in the following.

Inheritance mechanism in the MOOD system uses copy semantics, i.e., attributes of the super classes are copied to the subclass with method definitions. This is used in C++, and the MOOD system is forced to use this approach to work with C++ defined classes.

In the inherited clause of the system, the order of the inherited class names is important since they are copied to the current class definition in that order.

The heuristic implemented to solve the name conflicts (shown in Figure 3.7a) is as follows: search the class's own attributes first, and if the name does not exist, search first inherited class (in Figure 3.7a Object A is searched) if found then use that name otherwise search for the next class and continue this way. With the rename facility of the system the user will be able to rename attribute names in order to solve name conflicts.

Precedence conflict is solved in the same manner, inherited classes in Figure 3.7b are in the order of [Object D, Object E, Object F , Object G, Object D, Object H and Object I] because of copy semantic mentioned earlier. By using heuristic defined above, search condition will find Object D's X attribute, which is in the first position in the class definition, and uses that one.

Set, List, Ref and Tuple constructors support the relations between objects of MOOD Kernel. These classes are system defined, which have some common attributes and methods available. Set and List store OID's of objects. User can define a Set or a List for class or instance groupings. These constructors can be applied recursively with Tuple constructor to define new class relations. As an example:

```
MOODSQL>            create class DEPARTMENT
               TUPLE ( NAME string[20],
                    FLOOR integer );
MOODSQL>            create class EMPLOYEE
               TUPLE ( NAME string[20],
                    AGE  integer,
                    POSITION string[20],
                    DEPT REF ( DEPARTMENT ) );
```

In the above example a class named DEPARTMENT is created and referenced from the definition of class EMPLOYEE through DEPT attribute. In the above example each EMPLOYEE instance points to a department instance which contains related department information.

Set property of the MOOD data model is that, membership to a set is implicitly defined by SQL queries or views. Set objects can not be explicitly defined by the user. With this approach there is no set hierarchy concept defined in the data model of MOOD. As an example:

```
MOODSQL> create class WHO_WHERE
    TUPLE ( DEPT REF(DEPARTMENT) ,
                    WORKER SET( EMPLOYEE ) );
```

With the above example a reference to DEPARTMENT class is created which points to a department instance (i.e., each instance of WHO_WHERE class has a pointer to a DEPARTMENT instance), and a set of EMPLOYEE instances are pointed. As can be seen, Set construction handles uniform objects (i.e., EMPLOYEE instances) so the definition of Set is homogeneous in the MOOD data model.

4.3.5 Aggregates in MOOD

Aggregate definitions are handled in MOOD system by introducing type constructors (Set, List, Ref and Tuple). Aggregate classes can be constructed by recursive use of above type constructors. Since copy semantics is used by the kernel in inheriting from the super classes, a newly created aggregate class does not need the class definitions used in the aggregation after creation. But since classes created in the form of aggregates point to the object of the classes, deletion of classes which are used in the aggregate construction, forces the system to delete instances and also the classes which use definitions of the deleted class. This problem will be avoided by introducing versioning to the MOOD system in the future.

In section 4.3.4, the last two examples given are, infact aggregate definitions. For example, WHO_WHERE class is created by using DEPARTMENT and EMPLOYEE class definitions.

A class can also be defined by recursively applying above constructors. For example:

```
MOODSQL> create class Composite
    TUPLE ( TUPLE ( composite_id integer ,
                    composite_name string[10]),
          TUPLE ( LINK REF ( EMPLOYEE ) ,
                SET_LINK SET ( DEPARTMENT ) ,
                TUPLE (LINK_ID  integer,
                    LINK_NAME string[20])
                    )
                )
          );
```

A network like aggregation concept is proposed, which does not have redundancy, because object identifiers are used in referring related instances. In Figure 3.9, the aggregation classification of MOOD system is given.

In Figure 3.10 integrated view of object abstraction roles is given. When a class is created in MOOD, instances of that class can be created at any time. But an instance can not be related with more than one class. In MOOD , meta-data means class definition and data implies the objects of the class. Hence, meta-data and data are distinguished. Although an instance can be a member of undefined number of sets, lists or could be referenced from other objects, there is no abstraction mechanism to identify object itself if it is used in those definitions.

4.4 Design of Catalog Manager(CM)

A CM is designed for handling data definitions. The responsibility of CM is to handle all type of definitions in the system. Shortly, CM is responsible from definition and use of MOOD data model whose characteristics are defined in section 4.3.

MOOD Kernel introduces three classes to store data definitions in the system.

- MoodsTypes ,
- MoodsAttributes ,
- MoodsFunctions.

MoodsTypes class instances keep definitions of classes, indexes and types of the database. Instances of MoodsTypes class can have pointers to MoodsAttributes class instances, which constitutes the attributes of the class definitions.

MoodsFunctions class instances keep track of the member function definitions of classes. The details of the information kept in MoodsFunctions class is described in the design and implementation of Dynamic Function Linker(DFL). The only purpose of this class is to support dynamic definition and linking of member functions of classes. C++ preprocessor(Cfront) is modified to extract needed information to construct instances of MoodsFunctions.

Basic data types(integer, float, string etc.) and type constructors(i.e., set, list, ref, tuple) defined by the system, are instances of MoodsTypes class, which are defined externally while creating the database. Thus, it is clear that, a user can define any type and related methods to the database.

4.5 Implementation of MOOD Catalog Manager

Catalog Manager (CM), is the main part of MOOD kernel which stores schema information of the MOOD system. The type constructors Set, List, Ref and Tuple and the basic types are supported by the system.

With the help of a Database class, a number of databases can be built and used. Database class instances hold information concerning MoodsTypes file identifiers for the given database. The model of database class is given in Figure 4.1.

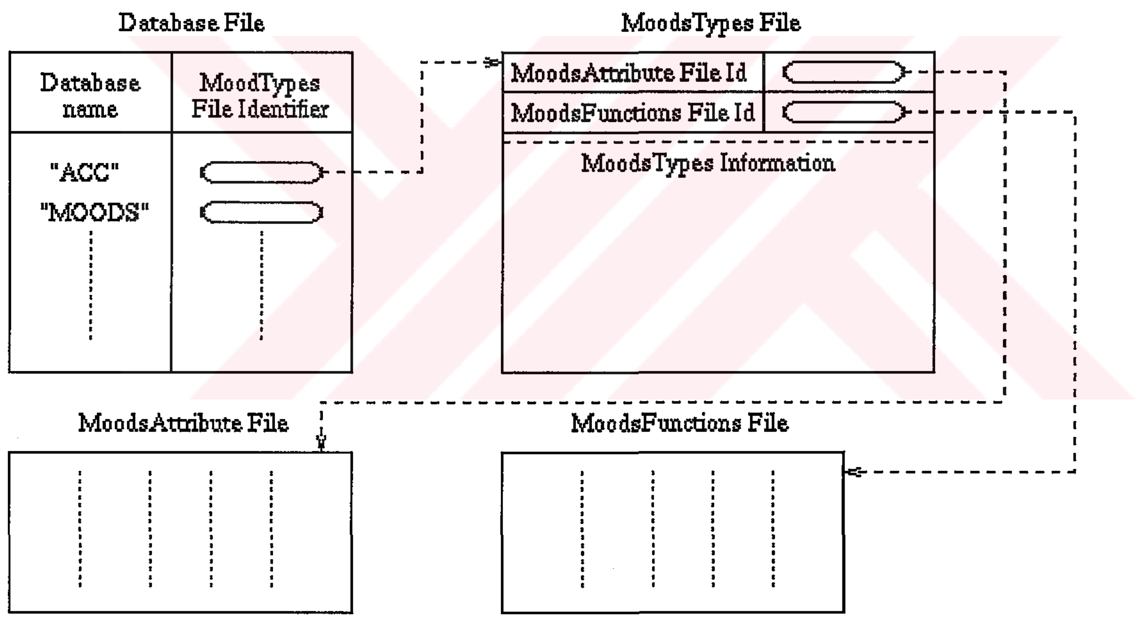


Figure 4.1 Database model

Database is opened by creating an instance of database class, and a file identifier (FID) is returned by the database class constructor, as an example consider the following C++ code segment.

```
{ BOOLEAN status;
  FID db_fid; // database FID will be stored.
  Database my_database("SCHOOL", &db_fid, &status);
};
```

In above example "SCHOOL" database opened, and FID of MoodsTypes file is retrieved for further operations on the database.

After obtaining database in use, an instance for MoodsAttributes is created by giving the FID received by the above code fragment. The created instance is used to carry out further database operations.

The basic structures (OID, Set and List) used in the implementation are depicted in Figure 4.2.

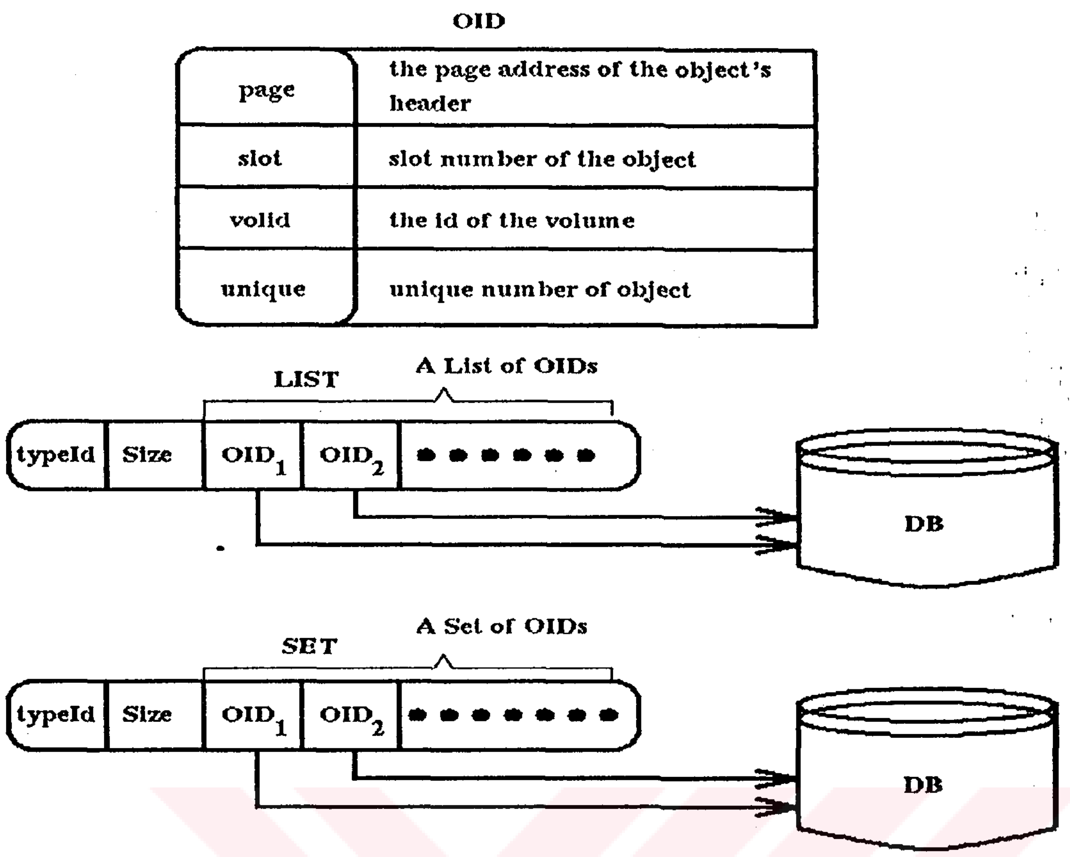


Figure 4.2 Structures of OID, Set and List

In the first two positions of the MoodsTypes file, the MoodsAttributes and MoodsFunctions file identifiers for the current database are stored. Using these file identifiers, the constructor of MoodsTypes class opens the related files. In the logical structure of MoodsTypes file in Figure 4.3, file identifiers are not shown since they are not relevant for the catalog data structures.

Database class of the MOOD system supports creation and deletion of databases in the kernel. If a given database does not exist, status variable returns a non-zero value, and a new database can be created by using "my_database.create_db()" call to database class. A database can also be deleted after creating instance by a call like "my_database.del_db()".

Creation of MoodsTypes instance is given in the below C++ code fragment:

```
{
{
 FID moods_fid;
 BOOLEAN status;
 Database my_database("SCHOOL",&moods_fid,&status);
 if ( status ) { /* exit, since no database with
given name*/
     cout << "no database found\n" ;
     exit(1);
 };

 MoodsTypes Catalog(moods_fid, &status);
 if ( status ) { // no way . };
 cout << "database and MoodsTypes are in use\n" ;
};
```

A description of MoodsTypes and MoodsAttributes classes with the relationship between them is depicted in Figure 4.3.

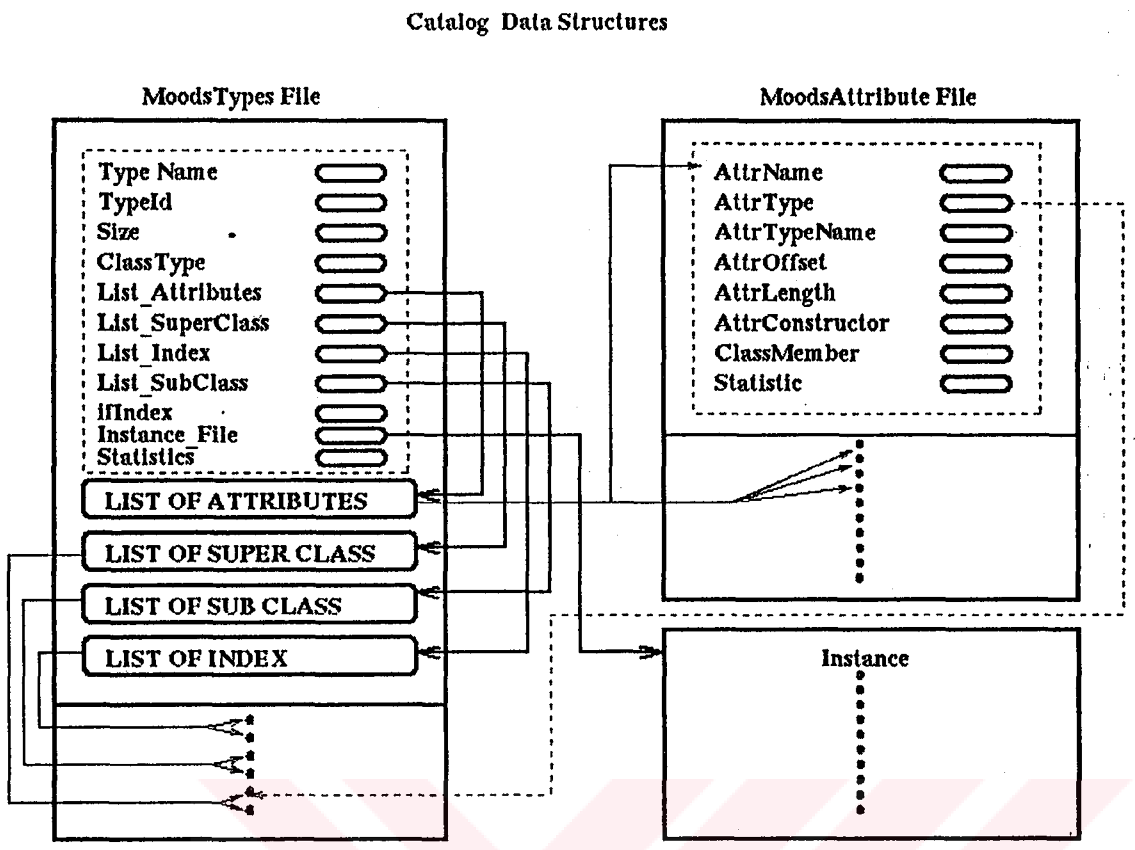


Figure 4.3 MoodsTypes and MoodsAttributes class structures

MoodsTypes data structure contains the information related with class, type and index structures(i.e., schema). Some of the information points to OID's of related objects in MoodsAttributes file. List structures are used for realizing 1:n relations to be established between MoodsTypes and MoodsAttributes objects.

The first attribute of MoodsTypes that is TypeName holds the name of the entry (class, type or index) defined in the system. TypeId is the unique type identifier, given by the Sequence class of the system at creation time of database types. The storage size of the defined entry is stored in Size field. The type of the entry(user defined class, system defined class, user defined type, system defined type, user defined index, etc.) is stored in ClassType field. The attributes' OIDs are collected to form a list as shown in Figure 4.2, and this list is pointed by List_Attributes (OID of the constructed list) field of MoodsTypes. Super classes' OIDs are collected in a list of super classes and pointed from MoodsTypes by List_SuperClass, similarly sub classes and indexes are pointed by List_SubClass and List_Index in that order. The ifIndex field contains the number of indexes created for the given entry. Instances created for a class are stored in a file, whose FID is kept in Instance_File attribute. Statistics attribute contains OID of the statistics structure which is used by query optimizer of the system.

MoodsAttributes file contains attribute information related with the classes. Attribute name is kept in AttrName field, attribute type is a pointer to the MoodsTypes entry, which shows the type of the attribute and kept as OID. Attribute type name is stored in AttrTypeName to be used when a class is to be described. Note that if this entry does not exist then for each attribute of class a pointer should be followed to MoodsTypes to get type name of the attribute. AttrOffset is the storage position of the attribute in class description, used for interpreting attribute at run time with AttrLength, which is the length of the attribute. AttrConstructor is the type of the constructor from which attribute is derived (i.e., set, list, tuple). ClassMember field stores information, on whether the attribute is owned by the class or inherited from some other class in the inheritance hierarchy. Statistic field contains OID of statistical data structures.

MoodsTypes and MoodsAttributes classes hold a part of database schema of the system. MoodsFunctions class will be described in Chapter V.

CHAPTER V

DYNAMIC FUNCTION LINKER

5.1 Introduction to Dynamic Function Linker

A Dynamic Function Linker (DFL) is designed and implemented in order to handle dynamic schema changes in the system. Stand alone version of DFL is tested and debugged. However it is yet to be integrated with the MOOD prototype. There are some restrictions coming from design of the DFL which will be discussed in this Chapter.

5.2 C++ preprocessor (Cfront)

Cfront is a utility written with yacc and lex programs of unix operating system. The goal is to convert C++ source code to C source code. Almost every C++ compiler set contains a Cfront utility[31].

Cfront is modified such that catalog information is extracted from the C++ function definitions. Catalog information extracted is the name, parameter types, return value and container class name of functions defined by the user.

5.3 Shared Object concept of SunOs

SunOs operating system supports dlopen(path,mode), dlsym(handle,symbol), dlclose(handle) and dlerror() functions to add a shared object to a program's address space, to obtain the address bindings of symbols defined by such objects, and to remove such objects when their use is no longer required.

dlopen() provides access to the shared object in path, returning a descriptor that can be used for later references to the object in calls to dlsym() and dlclose(). If path was not in the address space prior to the call to dlopen() then it will be placed in the address space, and if it defines a function named _init that function will be called by dlopen(). If however, path has already been placed in the address space in a previous call to dlopen(), then it will not be added a second time, although a count of dlopen() operations on path will be maintained. Mode is an integer containing flags describing options to be applied to the opening and loading process, for the time being-always set to 1. A null pointer supplied for path is interpreted as "main" executable of the process. If dlopen() fails, it will return a null pointer.

dlsym() returns address binding of the symbol described in the null-terminated character string symbol as it occurs in the shared object identified by the handle. The symbols exported by objects added to the address space by dlopen() can be accessed only through calls to dlsym(), such symbols do not supersede any definition of those symbols already present in the address space when the object is loaded, nor are they available to satisfy "normal" dynamic linking references. dlsym() returns a null pointer if the symbol can not be found. A null pointer supplied as the value of handle is interpreted as a reference to the executable from which the call to dlsym() is being made (i.e., thus a shared object can reference its own symbols).

dlclose() deletes a reference to the shared object referenced by handle. If the reference count drops to 0, and if the object referenced by handle defines a function _finit that function will be called, the object removed from the address space, and handle destroyed. If dlclose() is successful, it will return a value of 0, otherwise a non-zero value is returned.

dlerror() returns a null-terminated character string describing the last error that occurred during dlopen(), dlsym() or dlclose(). If no such error is encountered, then dlerror() will return a null pointer.

5.4 Dynamic Function Linker Design

The problem of getting a shared object file to the address space of the current process, and to find address space binded to a function is handled by the SunOs system calls described in section 5.3. But there are still the following problems to be solved:

- locating shared object file in which the given function resides,
- locating the given function's binding in the dynamically linked shared object,
- passing parameter values to the given function and
- receiving the return value of the function.

To locate shared object file of a function, all functions of the same class are kept in a file, whose name is derived from the class name. This information is stored in the CM.

MoodsFunction class has the information about the function signature created by C++ preprocessor (Cfront), which is searched in the shared object of class at run time by using dlsym() routine and the memory start address of the function is obtained(i.e., binding of function).

A uniform Function class is implemented, which handles dynamic linking of all functions defined in the system. This class does not have any information on the requested function, except for the parameter types, the return type, and the argument count, and the name of function. The function call is as follows:

```
{
void *(*dummy) (...) ;
dummy = (void *(*) (...)) (dlsym(handle,func_name))
;
};
```

The given function name is searched from the catalog, and when the function signature is found, it is used as parameter to dlsym() to locate binding of the function.

Parameter passing to functions is achieved by a Parameter class. On each call of a function, interpreter fills the structures in this class as shown in Figure 5.1.

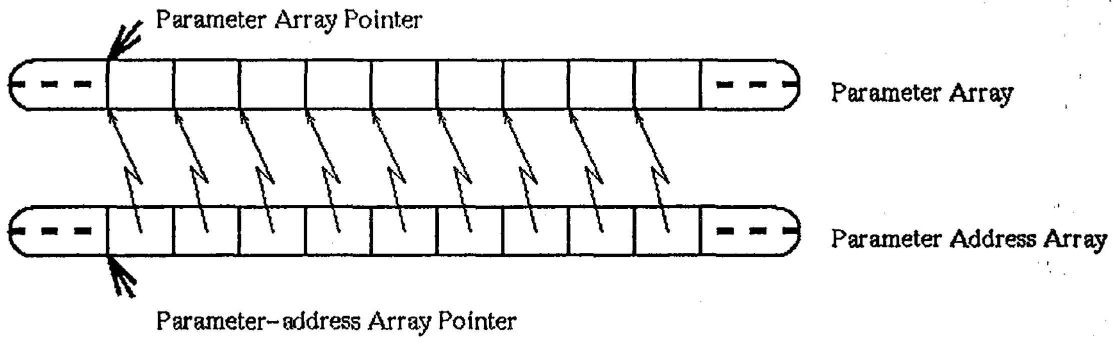


Figure 5.1 Parameter Class Definition

An instance of Parameter class is created whose parameter is the total storage size of the parameters. By using add_parameters function of the class, parameters are appended to the array. As an example:

```
{
 my_class param1;
 char *param2;

 Parameter my_par(24);
 my_par.add_parameters(&param1, sizeof(param1));
 my_par.add_parameters(&param2);

};
```

Pointers should be passed to the add_parameters function without a size parameter. Parameter address array is built by Parameter class, where each parameter's start offset in the parameter array is pointed by parameter address array. Constructed arrays can be received by get_parameters() and get_indexes() function calls. An instance of Function class is created with the constructed parameter and index arrays as follows:

```
{
 Function my_func(file_name,function_name,
                  my_par.get_parameters(),
                  my_par.get_indexes());
 if (my_func.error)
  ret_val = (my_type *)my_func.dynamic_link_func();
};
```

SQL interpreter, when needed obtains the file_name (i.e., class name) and function_name from the CM. However, if a call is made from C++ source, this information is extracted from the definition of the function and the class on which the function is applied.

With modified Cfront, function calls within function definitions are detected and are converted to Parameter and Function class calls. Return statement is replaced with moods_return, which copies the address of object to be returned to the global ret_val variable and returns ret_val from function. Recursive functions are not supported by the system.

When a user wants to define a new function to the system he/she writes the C++ code and through MOODSQL interpreter defines the function to the system. The function is passed from the modified Cfront which extracts needed information, makes the required changes on the function(i.e., all functions return types are changed with "void *"). Furthermore, the newly defined function is compared with the existing functions of the class to handle the name conflicts. Then, the function is compiled and linked with the owner class' shared object file. The header file of the class is also updated, which thus contains information about the new function.

Exception class is implemented for handling run-time crashes. If a user defined function has some semantic errors, the only way to get rid off them is to handle the system signals and walk over the function, and delete it from all places and rollback the database state.

CHAPTER VI

CONCLUSION

An Object Oriented Database Management System Kernel is designed and implemented. The kernel is based on ESM and includes the implementation of a Catalog manager and a dynamic function linker as described in Chapter III, the object-oriented concepts implemented through the MOOD kernel are compared with the previous works. The basic design strategy is to be able to use full power of a programming language(C++) in object definitions. Hence, some of the design decisions are necessitated to minimize the impedance mismatch with the C++ language.

The originality of proposed kernel structure is that, user can use full power of C++ language as well as SQL. A new dynamic schema change model, namely Dynamic Function Linker (DFL), is proposed which best fits the requirements of object oriented systems.

Although MOOD DFL is the fastest approach for handling dynamic schema changes, it has some problems which is yet to be solved. However, these problems stem from the user rather than the system.

First problem arises if a function is called within a function definition. In this case, if the function updates some objects, these updates may propagate throughout the system, without the control of the Kernel. However, handling this problem may be left as user's responsibility as mentioned in[32].

Second, there is no object update policy designed so far. Hence, it is the user's responsibility to take care of propagated object updates.

Since Kernel does not have control over user defined functions, erroneous user codes may have side-effects, and may damage the database.

Future work on DFL is to devise a mechanism, if possible, to find out the side-effects of the methods and introduce policies to control them.

As a conclusion DFL design and implementation may be the fastest that can be achieved. Also if used properly, it is in our opinion is the best solution handling the dynamic schema changes and late binding problems in object-oriented DBMSs.

REFERENCES

[1] J.R. Abrial , Data Semantics, in: Data Management Systems, (eds., J.W. Klimbie, K.L. offeman), North-Holland Publ. Comp., Amsterdam, (1974).

[2] D. Bobrow, A. Collins, Representation and Understanding, Academic Press, New York, (1975).

[3] A. Borgida, et. al., "Generalization/Specialization as a Basis for Software Specification", Topics in Information Systems, Spinger-Verlag, New York, (1986).

[4] N.M. Mattos, "Abstraction Concepts: the Basis for Data and Knowledge Modelling", ZRI Research Report No. 3/88, University of Kaiserslautern, 7th Int. Conf. on Entity-Relationship Approach, Rome, 331, (November. 1988).

[5] __________, Informix-Online Transaction Processing Database Engine for SQL Product for the UNIX Operating System Adminastrators Guide, Informix Software Inc., (1990).

[6] __________, ORACLE RDBMS Database Administrator's Guide, Oracle Corporation (1989).

[7] Cem Evrendilek, "Persistent C++ In DOS Environment: C*", M.S. Thesis in Computer Engineering, Middle East Technical University, Ankara, (1992).

[8] M.Cray, et. al., "Object and File Management in EXODUS Extensible Databases", Proc. of Int. Conf. on VLDB, (1986).

[9] A. Goldberg, D. Robson, Smalltalk-80 - The language and Its Implementation , Addison-Wesley Publ. Comp., Massachusetts, (1983).

[10] Bernhard M., et. al. , "Grand Tour of Concepts for Object-Orientation from a Database Point of View", Univ. of Kaiserslautern Computer Science Dept. Technical Report (1992).

[11] A.Borgida, "Survey of Conceptual Modelling of Information Systems", Proc. of handout to OODBTG Workshop, Ottawa, 461, (October, 1990).

[12] S.Khoshafian, G. Copeland, "Object Identity" , Proc. of OOPSLA (1986).

[13] T.W. Olle, The Codasyl Approach to Data Base Management, Wiley & Sons, London, (1978).

[14] C.J. Date, An Introduction to Database Systems, Addison-Wesley Publishing Company, Mass., (1983).

[15] P. Dadam, et al. "A DBMS Prototype to Support NF2-Relations" An Integrated View of Flat Tables and Hierarchies, in: Proc. ACM SIGMOD Conf., Washington, D.C., 356 , (1986).

[16] H.B. Paul, H.J. Schek, et al "Architecture and Implementation of Darmstadt Database Kernel System", Proc. of ACM SIGMOD Conf., San Fransisco, 196, (1987).

[17] B.Meyer, Object-Oriented Software Construction, International Series in Computer Science, Prentice Hall, NewYork, (1988).

[18] __________, ONTOS Developer's Guide, Ontologic inc.Version 2.01, Billerica, Mass., (May 1991).

[19] B. Stroustrup, "An Overview of C++", SIGPLAN Notices, 21(10), 525 (1986).

[20] V. Linnemann, et. al. "Design and Implementation of an Extensible Database Management System Supporting User Defined Data Types and Functions", Proc. of the VLDB, Los Angeles, 294, (1988).

[21] O. Deux, et. al. "The O2 System", Communications of the ACM, 34(10), 34, (1991).

[22] L.A. Rowe, M. Stonebraker, "The Design of POSTGRES" , Proc. of ACM SIGMOD Conf., Washington, D.C., 340, (1986).

[23] W.Kim, et. al. "Features of the ORION Object-Oriented Database System", Object-Oriented Concepts, Applications, and Databases, Addison-Wesley Publ. Comp., London (1989).

[24] N.M. Mattos, "KRISYS- A Multi-Layered Prototype KBMS Supporting Knowledge Independence", ZRI Research Report No. 2/88, University of Kaiserslautern, in: Proc. Int. Computer Science Conference - Artificial Intelligence: Theory and Application, Hong Kong, 31, (December, 1988).

[25] D.H. Fishman, et. al. "Overview of the Iris DBMS", Object-Oriented Concepts, Applications, and Databases, Addison-Wesley Publ. Comp., London (1989).

[26] C. Mohan, et. al. " ARIES: A Transaction Recovery Method Supporting Fine-Granularity Locking and Partial Rollbacks Using Write-Ahead Logging," IBM Research Report RJ6649 (January, 1989).

[27] J.N. Gray, "Notes on Database Operating Systems", Lecture Notes in Computer Science 60, Advanced course on Operating Systems, ed. G.Seegmuller, Springer Verlag, New York (1978).

[28] M. Chou, D. Dewitt, " An Evaluation of Buffer Management Strategies for Relational Database Systems," Proc. of the VLDB Conf., Stockholm, Sweden, (August, 1985).

[29] J. Orenstein, E. Bonte, "The Need for a DML: Why a library interface isn't enough" , Proc. handout to OODBTG Workshop, Ottawa, 83, (1990)

[30] D. Maier, J. Stain, "Development and Implementation of an Object-Oriented DBMS", Proc. of Int. Conf. on Object Oriented Programming, Systems, Languages and Applications, 167,(1986).

[31] __________, SPARCompiler C++, Language System Product Reference Manual, SunPro inc. (1992)

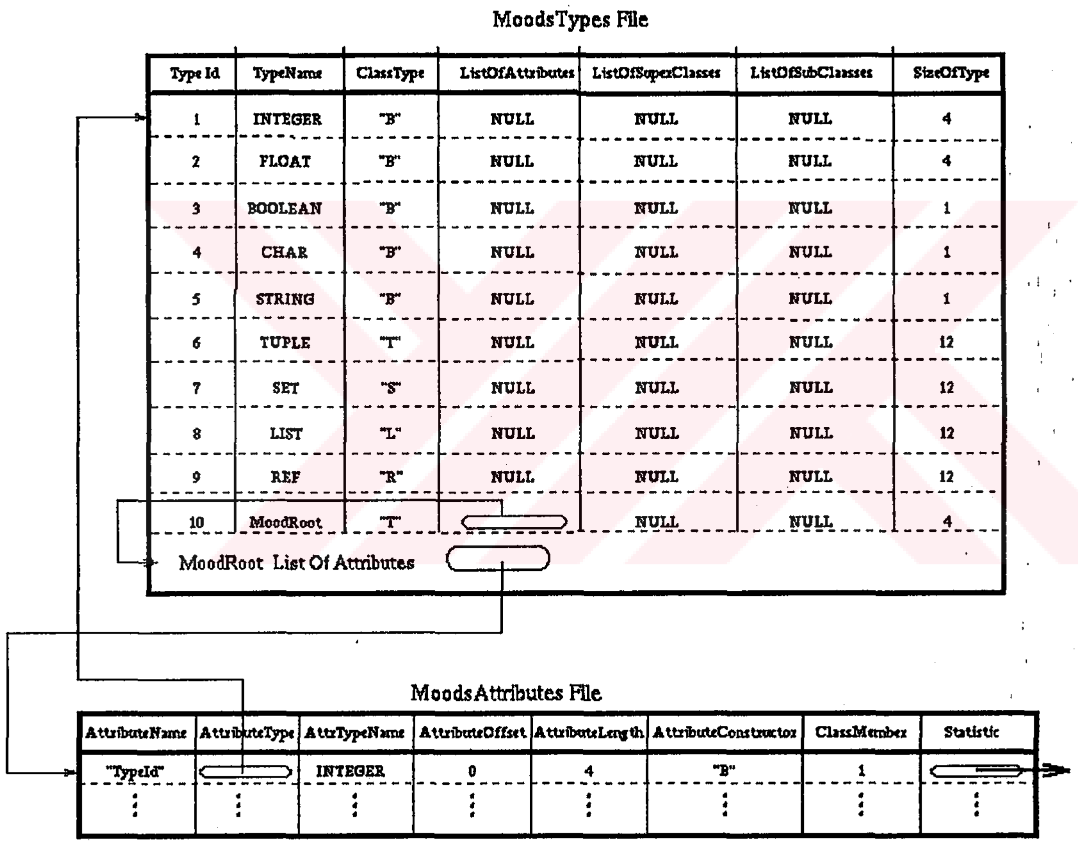
[32] K.J. Lieberherr, I.M. Holland, "Assuring Good Style for Object-Oriented Programs", IEEE Software, 38, (1989).

APPENDICES

APPENDIX A

CATALOG DATA STRUCTURES

In the following section, data structures used by catalog manager are described in detail.



MoodsTypes File

TypeId	TypeName	ClassType	ListOfAttributes	ListOfSuperClasses	ListOfSubClasses	SizeOfType
1	INTEGER	"B"	NULL	NULL	NULL	4
2	FLOAT	"B"	NULL	NULL	NULL	4
3	BOOLEAN	"B"	NULL	NULL	NULL	1
4	CHAR	"B"	NULL	NULL	NULL	1
5	STRING	"B"	NULL	NULL	NULL	1
6	TUPLE	"T"	NULL	NULL	NULL	12
7	SET	"S"	NULL	NULL	NULL	12
8	LIST	"L"	NULL	NULL	NULL	12
9	REF	"R"	NULL	NULL	NULL	12
10	MoodRoot	"T"		NULL	NULL	4

MoodRoot List Of Attributes

MoodsAttributes File

AttributeName	AttributeType	AttrTypeName	AttributeOffset	AttributeLength	AttributeConstructor	ClassMember	Statistic
"TypeId"		INTEGER	0	4	"B"	1	
⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮

Figure A.1 Initial Database Structures

Figure A.1 describes the initial database structures stored on ESM. First five type definitions are INTEGER, FLOAT, BOOLEAN, CHAR, and STRING with

TypeId fields 1 to 5. Next, constructors defined by the system are stored in the order TUPLE, SET, LIST, REF, which have type constructors "T","S","L","R". ListofAttribute, ListofSuperClass and ListofSubClass fields are empty for all of these types.

MoodRoot class definition is the last definition of the initial database, which is the superclass of each class in the system. MoodRoot class type entry has a OID stored for its attribute list, by following this OID, attribute List of MoodRoot class is reached. This list contains the OID of attributes in the MoodAttribute file.

The only attribute of MoodRoot class is TypeId and defined as an INTEGER. Every attribute entry in the system has some basic fields which are AttributeName (name of the attribute), AttributeType (refers to the class entry in the MoodsTypes file), AttributeTypeName (holds the name of the derived type), AttributeOffset (keeps the position of the attribute entry in the class definition), AttributeLength (which is used with AttributeOffset by the run-time system to interpret attribute), AttributeConstructor (which is used in showing whether the attribute is a constructor that refers to another type structure), ClassMember field (which is used in show if the attribute is owned by the class or inherited from another class in the system), and statistics field (contains an OID of the statistic object which will be implemented in near future).

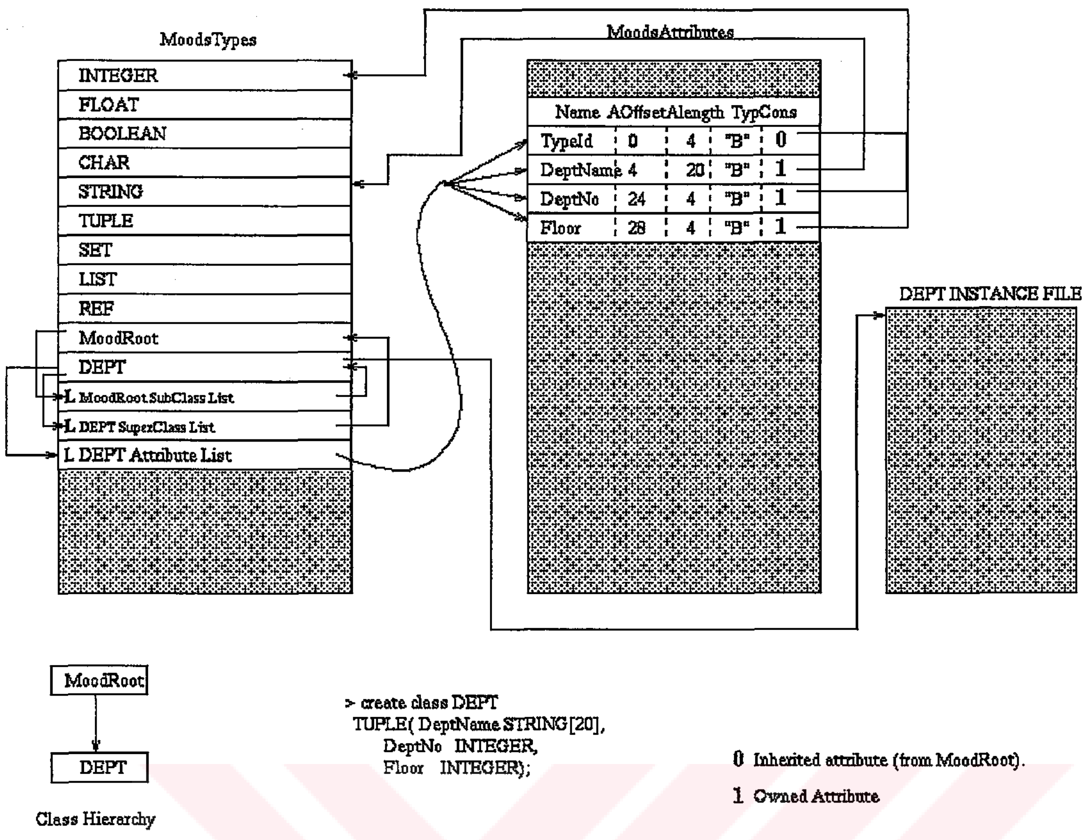
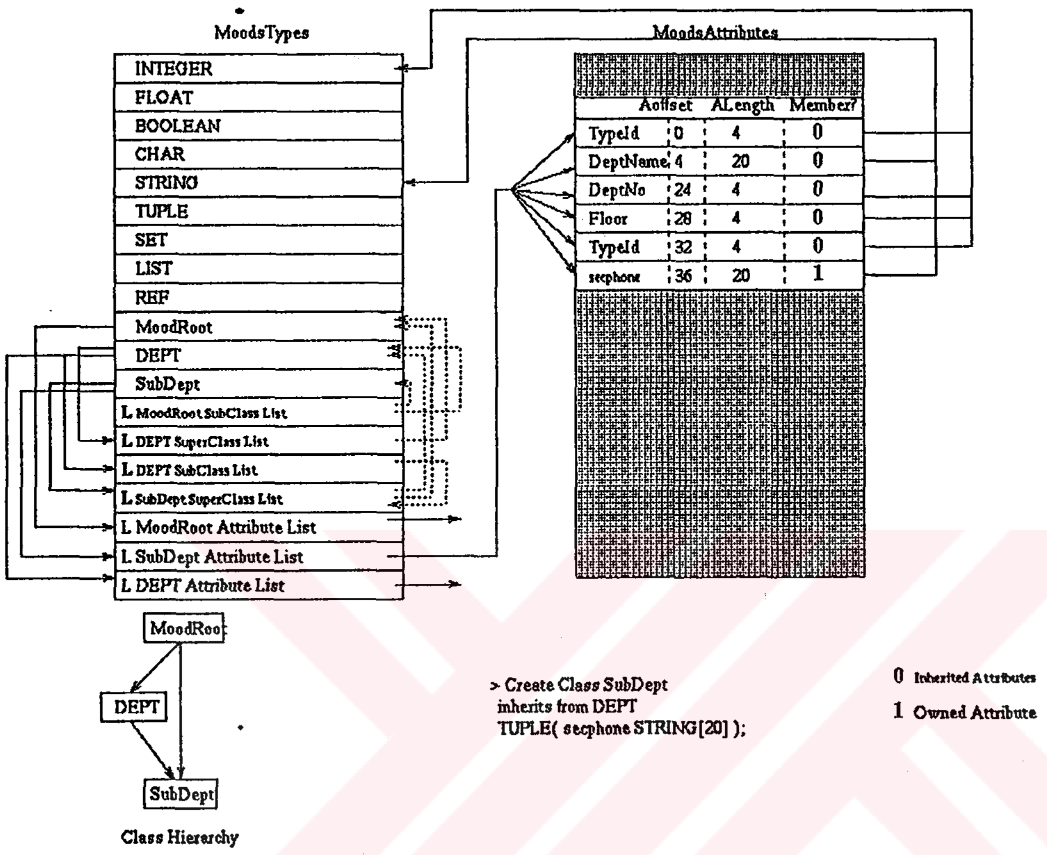


Figure A.2 Definition of a class in the system

In the Figure A.2 definition of DEPT class to the system is illustrated in a simple way. The data definition language of MOODSQL accepts class definitions in the syntax shown in the lower center of the Figure A.2. DEPT class contains three attributes namely DeptName, DeptNo and Floor which are of the types STRING, INTEGER and INTEGER respectively. In the DEPT definition of MoodsTypes there are two lists formed. First list contains superclass OIDs of the DEPT class, while the second list is attribute list which refer to the attributes of the DEPT class. Attribute list of DEPT class holds four OID values which refer to the above attributes plus TypeId attribute inherited from MoodRoot. With the definition of the DEPT class MoodRoot class's subclass attribute list is updated to hold the OID of the DEPT class definition

definition entry. Class hierarchy of the current state of the database is shown on the lower left part of the Figure A.2.



```
> Create Class SubDept
  inherits from DEPT
  TUPLE( secphone STRING[20] );
```

Figure A.3 Inheritance schema of the system

Inheritance information is stored in the system as shown in Figure A.3. In the Figure A.3 a new class SubDept is defined to the system with the "inherits from" clause, which inherits from DEPT class. As mentioned before inheritance mechanism copies attributes of the inherited classes to the defined class. TypeId, DeptName, DeptNo and Floor are copied from DEPT class and marked as zero on the Figure A.3. Notice that, SubDept class should has a unique TypeId field which is inherited from MoodRoot comes next and finally the only attribute of the subdept class is stored. The Subclass and SuperClass lists are updated to hold new schema of the system.

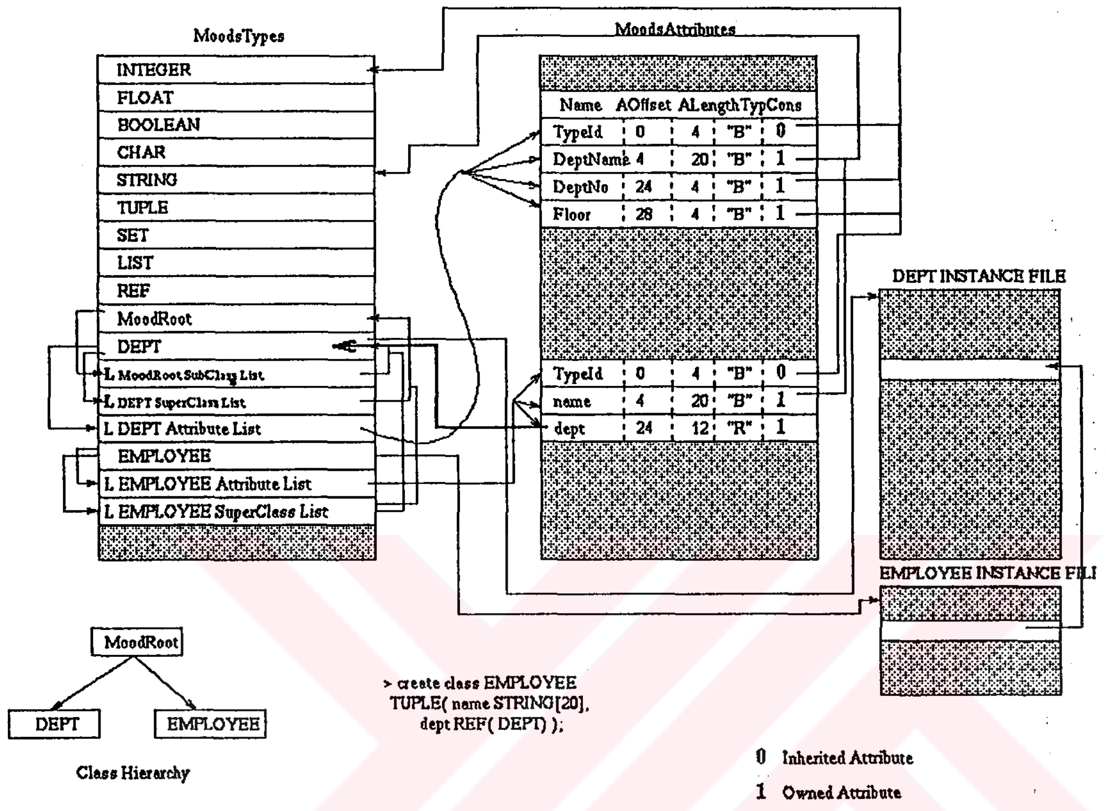


Figure A.4 REF structure of the system

In Figure A.4 the usage of REF type constructor is illustrated. Every instance of the EMPLOYEE class holds a REF field which holds the OID of an instance of the DEPT class. Since AttributeConstructor holds an "R" in the attribute entry of the EMPLOYEE class, system uses the OID stored at this position to receive the instance of the DEPT object. As shown in the Figure A.4, one pointer points to the definition of the DEPT class, which is used to interpret the DEPT instance read from the disk.

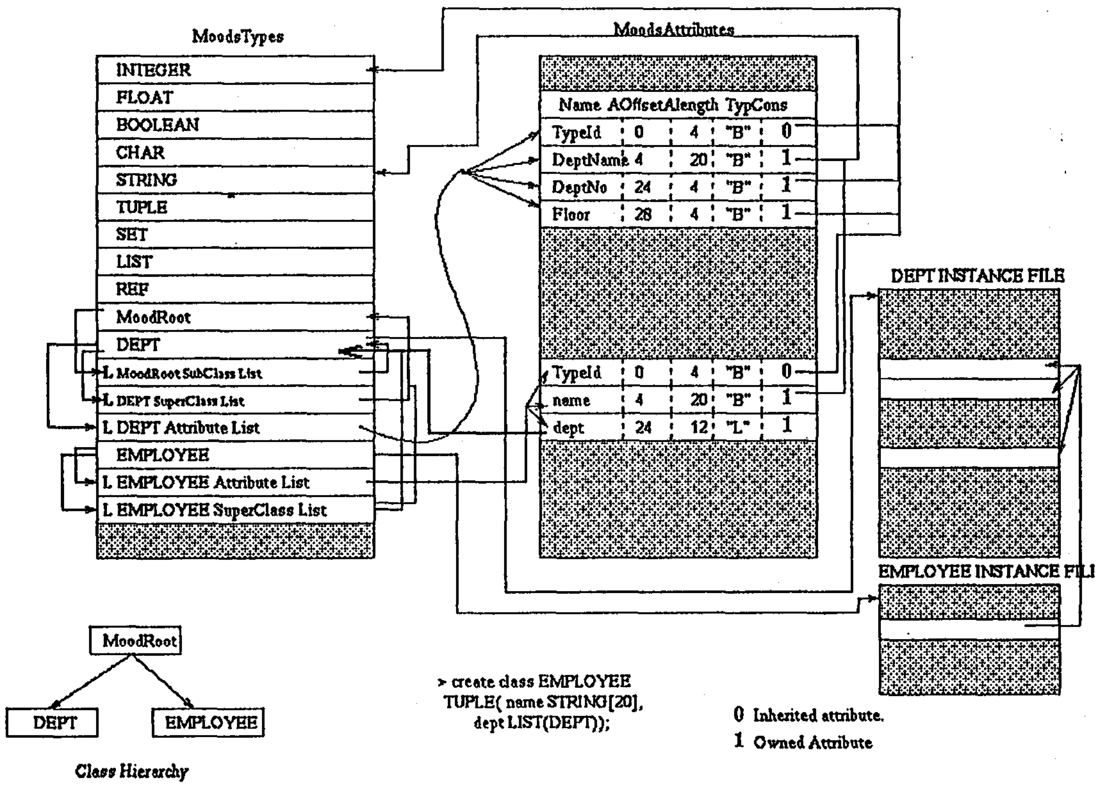
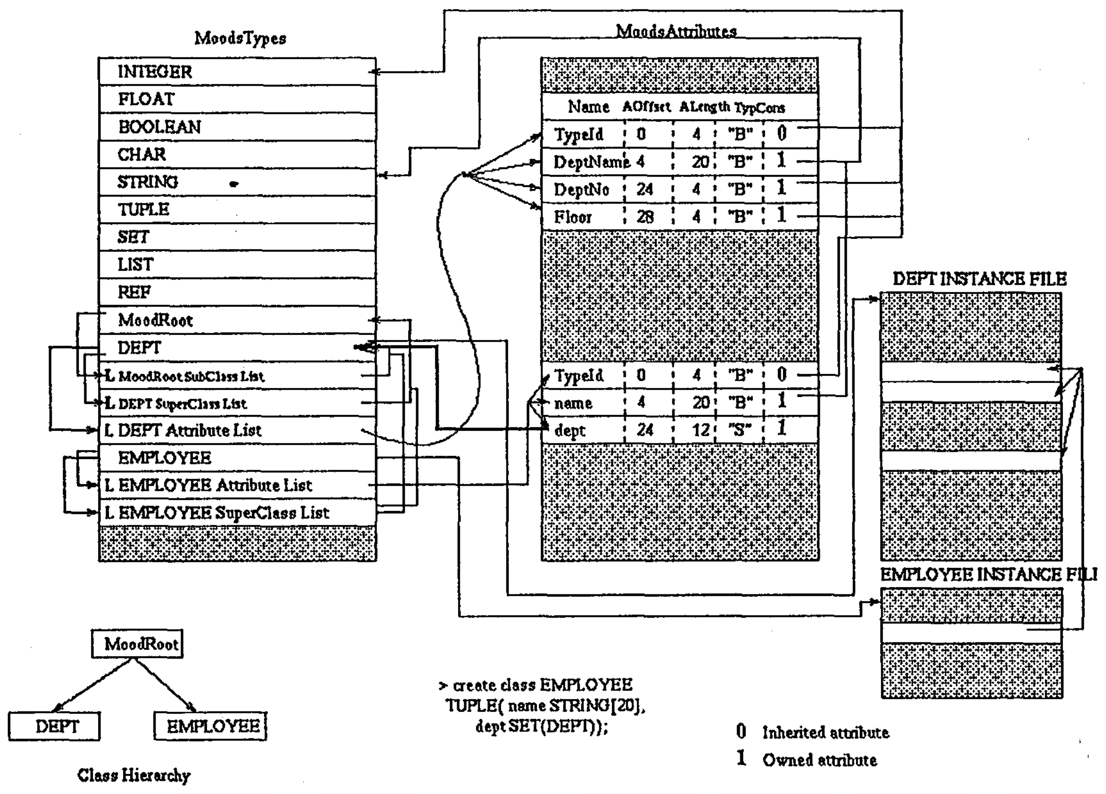


Figure A.5 LIST constructor of the system

LIST constructor is used to refer a group of instances of another class within the class defintion of a new class. List constructor has some operations like list_addition, list_subtraction, list_difference and elements of the list can be accessed like elements of an array. In Figure A.5, EMPLOYEE class has an OID field that refers to a list which holds the OIDs of instances of the DEPT class. The AttributeConstructor field (dept attribute) is interpreted by the system to find the list of OIDs of the instances of the EMPLOYEE class.

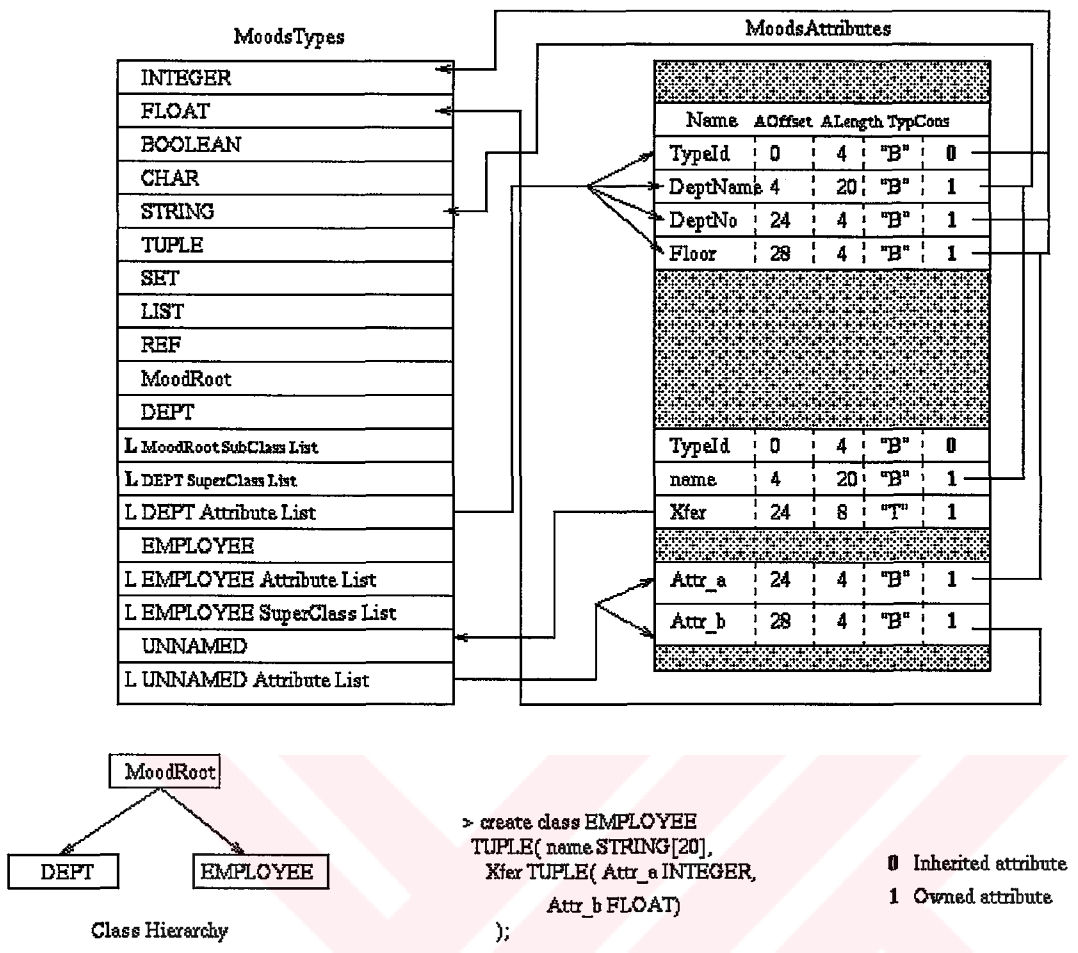
With the LIST definition, repetition of instances are allowed and the order of apperance is important.



```
> create class EMPLOYEE
  TUPLE( name STRING[20],
  dept SET(DEPT));
```

Figure A.6 SET constructor of the system

SET constructor of the system holds a set of OIDs, and has operations defined on sets such as union, intersection, iselement, and difference. Duplicate elements are not allowed. In Figure A.6, the definition of the class EMPLOYEE holds an attribute which is a SET. The dept field of EMPLOYEE class holds an OID value which is used to construct a set, hence the constructed set has OID values of some of the DEPT instances.



```
> create class EMPLOYEE
  TUPLE( name STRING[20],
    Xfer TUPLE( Attr_a INTEGER,
          Attr_b FLOAT)
  );
```

Figure A.7 Recursive Declerations

In Figure A.7 recursive decleration properties of the system is illustrated. A Tuple constructor whithin a Tuple constructor is used as an example. Recursive declerations are handled in such a way that in each level of recursive decleration an UNNAMED type is created to hold inner level of recursive information. In the example given in Figure A.7, a class EMPLOYEE is defined to the system which has a Xfer attribute which is also a tuple. An UNNAMED type entry is created and the AttributeType field is set to point to this entry. Since attribute type is known to be a tuple from AttrConstructor, the linkes are followed for the UNNAMED tuple to get the complete description of the attribute.

APPENDIX B

SOURCE CODE LISTINGS

```
#ifndef __ATTRIBUTE_H_
#define __ATTRIBUTE_H_
#endif

#ifndef __INCLUDE_ALL_
#define __INCLUDE_ALL_
#include <stdio.h>
#include <string.h>
#include <stdlib.h>
#include <setjmp.h>
#include "globdef.h"
#include <sm_client.h>
#include <stream.h> // used in Oid.h
#include <Oidh.h>
#include <esm_opsh.h>
#endif

extern  FILE  *sm_ErrorStream ;
extern  VOLID VolumeId ;
extern  TID   TransactionId;
extern  int   BufferGroup;
extern  FID   FileIdentifier;
extern  OBJHDR ObjectHeader;
extern  FILE  *ErrStrm;
extern  ERTYPE ErrorCheck(int e,char *routine) ;

#ifndef  MAXSTRLEN
#define  MAXSTRLEN 100
#endif
#define  MAXATTR   100

// this definition is needed for exodus storage manager....

typedef struct {
        char    AttributeName[MAXSTRLEN];
        OID     AttributeType;
        char    AttrTypeName[MAXSTRLEN];
        int     AttributeOffset;
        int     AttributeLength;
        char    AttributeConstructor;
        char    ClassMember;
        OID     Statistic;
} AttributeType;

typedef struct Alist {
        AttributeType entry;
        Alist       *next;
        Alist       *nextsub;
} AttrList;

class CatATTRIBUTE {
      public:
            USERDESC *UserDesc ;
            FID      AttributeFid;

            CatATTRIBUTE(FID ,OID *); // Constructor for Attribute..
            CatATTRIBUTE(FID);        // Constructor for Attribute..
            void writeAttr(AttributeType ,OID *);
            void readAttr(OID ,AttributeType *, CatBOOLEAN *);
            void disp_attr(AttributeType );
                                void updateAttr( OID, AttributeType );
};
```

catalog.h

```
#include <classddl.h>
#ifndef  CatBOOLEAN
#define  CatBOOLEAN char
#endif

#ifndef __ATTRIBUTE_H_
#include <attribute.h>
#endif

#include <lists.h>
#include <sequence.h>

#define SEQNAME       "$MoodsType"
#define SYS           0
#define USER          1
#ifndef MAXSTRLEN
#define MAXSTRLEN     100
#endif
#define MAXINHERITS   40
#define MAXCLASS      100
#ifndef S_OID
#define S_OID         sizeof(OID)
#endif
#ifndef S_FID
#define S_FID         sizeof(FID)
#endif
#define S_CATALOG     sizeof(Catalog)
// define built-in types ******
#define TYPE_SET      7
#define TYPE_LIST     8
#define TYPE_REF      9
#define TYPE_TUPLE    6
// *********** end .....

OID     nullOid;
FID     nullFid;
OID     rootOid;

typedef struct {
        char TypeName[MAXSTRLEN];
        int  TypeId;
        int  Size;
        char ClassType; // 'C' system class , 'c' user class,
                                 // 'T' system type, 't' user type
                                       // 'B' basic type....
   OID  List_Attributes;
        OID   List_SuperClass;
   OID  List_SubClass;
        OID   List_Index;
        char  ifIndex; // 0 for no index 1 for index ;
   FID  Instance_File;
   OID  Statistics;
} Catalog;

typedef struct levelstruct {
        char  ClassName[MAXSTRLEN] ;
        levelstruct *next ;
} LevelInfo;

typedef struct hinfo {
        LevelInfo *thislevel;
        hinfo      *nextlevel;
} Hierarchy;
```

```
typedef struct superClassName {
                   char Name[MAXSTRLEN] ;
                   superClassName *next ;
} ClassNames ;


typedef struct {
        FID  FileAttr;
        FID  FileFunc;
 } Internal_Files;

class MoodsTypes{
      private:

           FID  CatalogId ;
           FID  AttributeId ;
           FID  FunctionId ;
           FID  InstanceId;

      public :
           USERDESC *UserDesc ;
           char typeName[MAXSTRLEN];
           int  typeId;
           int  typeSize;
           int  classVariables;
           char classType;  //   System class | User Class | User typ
                            //   | Basic typ | Sys typ
           char constructor; //  Tuple,set,list,ref,noconstructor
           OID  attributes;  //  Attribute Oid's
           OID  subClasses;  //
           OID  superClasses;//

                MoodsTypes(FID );
                MoodsTypes();
           void addType( char *, LinkedList *,  memberNode *, int , CatBOOLEAN *);
           void MoodsAttr( memberNode *, int *, List *, CatBOOLEAN *) ;
           void createUnnamedType( memberNode *, int *, OID *, CatBOOLEAN *);
           void setClassType( char *, int );
           void getTypOid( char *, OID *, CatBOOLEAN *);
           void getTypOid( int    , OID *, CatBOOLEAN *);
           void copyInheritedTypes( OID *, List *, int *, CatBOOLEAN * );
           int  findType( char * );
           void createInstanceFile( FID *, CatBOOLEAN *);
           void linkToSuper( OID, OID );
           void readType( OID, Catalog *, CatBOOLEAN *);
           void readType( char *, Catalog *, CatBOOLEAN * );
           void writeType( Catalog, OID *, CatBOOLEAN * );
           void deleteType( OID ) ;
           void dropFromSuperClass( OID , OID , CatBOOLEAN *);
           void deleteClass( OID , int *, CatBOOLEAN *) ;
           void deleteClass( char * , int *, CatBOOLEAN *) ;
           void updateType( OID , Catalog ) ;
           void deleteAttributes( OID , CatBOOLEAN *);
           void deleteFiles();
           void getAttrFile( FID * );
           void describe (OID, int = 0 );
           void describe ( char * );
           void printvals(Catalog );
           void dispTypes();

           void getClassInfo(char *, Catalog *, AttrList**, CatBOOLEAN*);
           void getClassInfo(char *, Catalog *, AttrList**,
        void getAttrInfo( OID, AttrList**, CatBOOLEAN*);
        void getClassHierarchy(char *,Hierarchy **,CatBOOLEAN *);
        void changeNameOfClass(char *, char *);
        void addAttributeToClass(char *, AttributeType, CatBOOLEAN *);
        void addAttributeToClass(char *, memberNode *, CatBOOLEAN *);
                        void printAllAtt(AttrList* );
                        void dispLevel( LevelInfo *);
                        void dispHier( Hierarchy *);
                        void dispSuperClass( char *) ;
                        void dispSubClass( char *) ;
                        void getSubClass( char *, ClassNames ** );
                        void getSuperClass( char *, ClassNames **) ;
                        void createIndex( char *, OID , CatBOOLEAN * );
                        void dropIndex( char *, OID, CatBOOLEAN *);

        // MoodsView functions.........

};
```

```
// external definitions
#ifndef __INCLUDE_ALL_
#define __INCLUDE_ALL_
#include <stdio.h>
#include <string.h>
#include <stdlib.h>
#include <setjmp.h>
#include "globdef.h"
#include <sm_client.h>
#include <iostream.h> // used in Oid.h
#include <Oidh.h>
#include <esm_opsh.h>
#endif
// external definitions
extern  FILE  *sm_ErrorStream ;
extern  VOLID VolumeId ;
extern  TID   TransactionId;
extern  int   BufferGroup;
extern  FID   FileIdentifier;
extern  OBJHDR ObjectHeader;
extern  FILE  *ErrStrm;
extern  ERTYPE ErrorCheck(int e,char *routine) ;


#define MAXDBNAME    100

class Database {
      public :
        char dbname[MAXDBNAME];
        FID  FCfile;     //  associated catalog file .
                         //  get or create db entry . see definition !!
        Database(char * , FID * , CatBOOLEAN *);
                        // delete db & assoc. information see definiton !!
        void deleteDB(CatBOOLEAN *) ;

};
```

```
#define DATABASE "$HOODS"
#define SEQNAME  "$HoodsType"
#define BASICMAX 20

#define SEQ_INITAL  0
#define SEQ_MIN     0
#define SEQ_MAX     2000      // 2000 types are allowed at the moment
#define SEQ_CYC     'N'
#define SEQ_INC     1

#include <catalog.h>
#include <database.h>

OID Type_Store[BASICMAX]; // store basic types OID values ?
int basic_count = 0 ;

void init_db(CatBOOLEAN *) ;
```

```
#ifndef __INCLUDE_ALL_
#define __INCLUDE_ALL_
#include <stdio.h>
#include <string.h>
#include <stdlib.h>
#include <setjmp.h>
#include "globdef.h"
#include <sm_client.h>
#include <iostream.h> // used in Oid.h
#include <Oidh.h>
#include <esm_opsh.h>
#endif

// external definitions
extern  FILE  *sm_ErrorStream ;
extern  VOLID VolumeId ;
extern  TID   TransactionId;
extern  int   BufferGroup;
extern  FID   FileIdentifier;
extern  OBJHDR ObjectHeader;
extern  FILE  *ErrStrm;
extern  ERTYPE ErrorCheck(int e,char *routine) ;
extern  int Test_Oid( OID Pfirst , OID Psecond , int Pcondition ) ;
extern  void display_Oid(OID oid) ;
extern  void read_Oid(OID *oid) ;


#define  MOODSLIST  135 // Moods List Type Classifier ....


  These definitions will be changed .....

#define  L_EXISTS  12345  /* OID value user wants to insert , already  there */
#define  NOTEXISTS 12346 /* OID value user wants to delete , not in the list */

/*     These definitions will be removed during entegration */

#define  F_NAME  "trial"

/*     Until This point .... */


//  ERTYPE return type means the error value of function ....
//  If an error (ESM or Logical) occurs return value is Error value
//  Otherwise Set to 0........


class MoodsRoot {
    public:
      int TypeId;
};


class List : public MoodsRoot {
      private:
        OID         ListOid;     // Oid of list object ......................
        OID        *ListHead;   // OID members start offset is pointed......
        void       *Buffer;     // Memory buffer to handle operations.......
        int         BufGrp;     // Buffer Group Universal variable is kept..
        TID         Tid;        // Transaction Id Universal variable is kept
        USERDESC   *UserDesc;   // User Descriptor for ESM .................
        FID         FileId;     // File ID (Given For Creation).............

  public:
    INT32       Size ;      // Size of current list......................
    INT32       ListPos;    // Current Position in list..................
    INT32       ListMax;    // Max size of list .........................

    List(OID *PListOid,TYPEID *PTypeId,SIZTYP *PSize,CatBOOLEAN *PStatus);
    List(FID *PFid,NTYPE PPos,OID *PNearOid,
         INT32 ListMax ,CatBOOLEAN *PStatus);
    OID   *operator[](INT32 Subscript);
    void   unpin() { if (UserDesc != NULL) {
                       sm_ReleaseObject(UserDesc);
                       UserDesc = NULL ;
                   };
                 } ;
    void   getNext(OID *,CatBOOLEAN *);
    void   getPrev(OID *,CatBOOLEAN *);
    ERTYPE updateList(OID *, INT32 , CatBOOLEAN *);
    ERTYPE insertToList(OID *,CatBOOLEAN *);
    ERTYPE insertToList(OID *,INT32 ,CatBOOLEAN *);
    ERTYPE deleteFromList(INT32 , INT32 , CatBOOLEAN *);
    ERTYPE isIn(OID ,CatBOOLEAN  *);
    ERTYPE list_Union(List ,List ,CatBOOLEAN *);
    void   intersect(List , List , CatBOOLEAN *);
    ERTYPE isSublist(List ,CatBOOLEAN *);
    ERTYPE isContain(List ,CatBOOLEAN *);
    OID    getListOid(CatBOOLEAN *) ;
    void   getListOid(OID *,CatBOOLEAN *) ;
    ERTYPE getElement(OID *,int ,CatBOOLEAN *)  ;
    ERTYPE getFirst(OID *,CatBOOLEAN *)  ;
    ERTYPE getLast(OID *,CatBOOLEAN *)  ;

};
```

```
// external definitions
#ifndef __INCLUDE_ALL_
#define __INCLUDE_ALL_
#include <stdio.h>
#include <string.h>
#include <stdlib.h>
#include <setjmp.h>
#include "globdef.h"
#include <sm_client.h>
#include <iostream.h> // used in Oid.h
#include <Oidh.h>
#include <esm_opsh.h>
#endif
// external definitions
extern  FILE  *sm_ErrorStream ;
extern  VOLID VolumeId ;
extern  TID   TransactionId;
extern  int   BufferGroup;
extern  FID   FileIdentifier;
extern  OBJHDR ObjectHeader;
extern  FILE  *ErrStrm;
extern  ERTYPE ErrorCheck(int e,char *routine) ;

#define SEQ_MAXNAME      20
#define SEQFILE          "$SEQFILE"
#define INCR_MULTIPLIER  5

typedef struct  {
       char seqname[SEQ_MAXNAME];
       int  minval ;
       int  maxval ;
       char cyclic ;
       int  initial;
       int  increment;
       int  curval ;
       OID  SeqId ;
} SeqTemp;


class Sequence {
           private :
               char seqname[SEQ_MAXNAME] ;
                int  minval ;
                int  maxval ;
                char cyclic ;
                int  initial;
                int  increment;
                int  curval ;
           public :
                OID  SeqId ;
           SeqTemp  SeqIntern ;
           USERDESC *UserDesc;
                Sequence(char *, int , int , int, char ,  int ,CatBOOLEAN *);
                Sequence(char *,CatBOOLEAN *);
           int  getNextVal(CatBOOLEAN *PStatus);
           int  getCurVal();
           void setCurVal(int ,CatBOOLEAN *);
           void setMaxVal(int ,CatBOOLEAN *);
           void setMinVal(int ,CatBOOLEAN *);
           void setCycOn();
           void setCycOff() ;
           void setInitial(int );
           void setIncr(int ,CatBOOLEAN *) ;
           void delSeq(CatBOOLEAN *);
};
```

```
#ifndef __SET_H
#define __SET_H

#include <stdio.h>
#include <strings.h>
#include <stdlib.h>
#include <setjmp.h>
#include "sm_client.h"

#define  MOODSSET  104

#define  BUFGRP   int
#define  NTYPE    int
#define  ERTYPE   int
#define  TYPEID   int
#define  SIZTYP   int
#define  GRPSIZE 200

// These definitions will be changed .....

#define  EXISTS   12345  /* OID value user wants to insert , already  there */
#define  NOTEXISTS 12346 /* OID value user wants to delete , not in the set */

/*    These definitions will be removed during entegration */

#define  F_NAME  "trial"

/*    Until This point .... */

#define  S_TypeId  sizeof(TypeId)
#define  S_OID     sizeof(OID)
#define  VOL       getenv("EVOLUME")


//  ERTYPE return type means the error value of function ....
//  If an error (ESM or Logical) occurs return value is Error value
//  Othewise Set to 0........


class Set : public MoodsRoot {
      private:
        OID        SetOid;    // Oid of set object .......................
        OID       *ListHead;  // OID members start offset is pointed......
        void      *Buffer;    // Memory buffer to handle operations.......
        int        BufGrp;    // Buffer Group Universal variable is kept..
        TID        Tid;       // Transaction Id Universal variable is kept
        USERDESC  *UserDesc;  // User Descriptor for ESM .................
        FID        FileId;    // File ID (Given For Creation).............


      public:
        int  Size ;           // Size of current set......................
        int        ListPos;   // Current Position in set..................

        Set(OID *PSetOid,TYPEID *PTypeId,SIZTYP *PSize,BOOLEAN *PStatus);
        Set(FID *PFid,NTYPE PPos,OID *PNearOid, BOOLEAN *PStatus);
        void   unpin() {sm_ReleaseObject(UserDesc);} ;
        void   getNext(OID *PEOid,BOOLEAN *PStatus);
        void   getPrev(OID *PEOid,BOOLEAN *PStatus);
        ERTYPE insertToSet(OID *POid,BOOLEAN *PStatus);
        ERTYPE deleteFromSet(OID POid,BOOLEAN *PStatus);
        ERTYPE isIn(OID POid,BOOLEAN  *PStatus);
        ERTYPE setUnion(Set PSet ,Set USet,BOOLEAN *PStatus);
        void   intersect(Set PSet , Set ISet , BOOLEAN *PStatus );
        ERTYPE isSubset(Set PSet,BOOLEAN *PStatus);
        ERTYPE isContain(Set PSet,BOOLEAN *PStatus);
        OID    getSetOid(BOOLEAN *PStatus) ;
        ERTYPE getElement(OID *POid ,int SetIndex,BOOLEAN *PStatus)  ;
        ERTYPE getFirst(OID *POid,BOOLEAN *PStatus) ;
        ERTYPE getLast(OID *POid,BOOLEAN *PStatus)  ;


};
/*
extern Set::Set(FID *PFid,NTYPE PPos,OID *PNearOid, BOOLEAN *PStatus);
extern ERTYPE Set::getFirst(OID *POid,BOOLEAN *PStatus);
extern ERTYPE Set::getLast(OID *POid,BOOLEAN *PStatus);
extern void   Set::getNext(OID *PEOid,BOOLEAN *PStatus);
extern void   Set::getPrev(OID *PEOid,BOOLEAN *PStatus);
*/

#endif
```

Params.h

```
#define MAX_PARAMS 5
```

```
#define F_MAX_C_NAME_LEN     40
#define F_MAX_F_NAME_LEN     40
#define F_MAX_FC_NAME_LEN    40
#define F_MAX_PARAMS         5
#define F_MAX_PATH_NAME      40
```

```
#include <lists.h>
#include <set.h>

VOLID VolumeId ;
TID   TransactionId;
int   BufferGroup;
FID   FileIdentifier;
OBJHDR ObjectHeader;
FILE  *ErrStrm;

ERTYPE ErrorCheck(int errcode, char *msg ); // prototype of error handling routine ....


//           ************** Set::Set ****************
//  Constructor for a known Set ( defined previously )
//  Just get the set in to memory if exists, otherwise
//  Set the PStatus variable to Error Code ....
//  IN  >>>     PSetOid   Set object's OID to get the object ...
//  OUT <<<     PTypeId   The Object's Type ID
//  OUT <<<     PSize     The Size of Object
//  OUT <<<     PStatus
//                        if 0 OUT ->set exists & loaded.
//                        if 1 OUT ->set does not exist.
//          ******************************************

Set::Set(OID *PSetOid,TYPEID *PTypeID,SIZTYP *PSize, BOOLEAN *PStatus) {
     // Copy Some important values to member variables ......

     int EsmError;

     BufGrp    = BufferGroup;
     Tid       = TransactionId;
     SetOid    = *PSetOid;
     FileId    = FileIdentifier;
     ListPos   = 0;

     //  Test if it Works !!!!!
     //  Read requested Set into memory....

     EsmError = sm_ReadObject(BufGrp,&SetOid,0,READ_ALL,&UserDesc);

     if(EsmError != esmNOERROR) {
        *PStatus   = EsmError;
        *PSize      = 0;
        PTypeID    = NULL;
        ErrorCheck(EsmError,"Set constructor, getting ...\n");
     } else {
        *PStatus=0;
        Size       = UserDesc->objectSize;
        Buffer     = UserDesc->basePtr ;
        ListHead   = (OID *) (UserDesc->basePtr + S_TypeId);
        PTypeID    = (int *)Buffer ;
        Size       = Size - S_TypeId;
        Size       = Size / S_OID;
        *PSize      = Size;
        // cout << "size of read set= " << Size << "\n" ;
     }
     TypeId = MOODSSET;
}

//           ************** Set::Set ****************
//  Constructor for Set creation
//  the Set is created which has only a TypeId Field filled with MOODSSET value.
//  Set the PStatus variable to Error Code ....
```

```
//  IN  >>>  PFid      File Id for the placement of the new object.
//  IN  >>>  PPos      Position dependency to given PNearOid.
//  OUT <<<  PStatus   returned from constructor . 0 for success, error# otherwise.
//           ****************************************

Set::Set(FID *PFid,NTYPE PPos,OID *PNearOid,BOOLEAN *PStatus) {
     // Copy Some important values to member variables ......

     int EsmError;
     void * buffer;
     TYPEID type  = MOODSSET;

     BufGrp    = BufferGroup;
     Tid       = TransactionId;

     // Create a Set object with no elements in file ................
     // Save   Set's Object Id in member variable for its lifetime...

     buffer =(void *) new TYPEID;
     memcpy(buffer,&type,S_TypeId);

     TypeId = MOODSSET;
     EsmError = sm_CreateObject(BufGrp,PFid,PPos,PNearOid,NULL, S_TypeId,buffer,&SetOid);


     if(EsmError != esmNOERROR) {
        *PStatus = EsmError ;
        ErrorCheck(EsmError,"creating set....\n");
     }
     else {
        *PStatus = 0;
       }
     Size = 0;
}

void Set::getNext(OID *POid ,BOOLEAN *PStatus) {

       if( (ListPos+1) < Size ) {
          memcpy(POid, &ListHead[ ++ListPos ],sizeof(OID));;
           *PStatus = 0;
       } else *PStatus = 1;
}

void Set::getPrev(OID *POid,BOOLEAN *PStatus) {

       if( ListPos > 0 ) {
         memcpy(POid , &ListHead[ --ListPos ],sizeof(OID));;
         *PStatus = 0;
       } else *PStatus = 1;
}


ERTYPE Set::insertToSet(OID *POid,BOOLEAN *PStatus) {

int ListIndex, Offset = 0;
int err;
OID *CurOid;

     BufGrp    = BufferGroup;
     Tid       = TransactionId;
     FileId    = FileIdentifier;

       // unpin();
```

```
    // Find Correct Place To Insert Oid .............

    if ( Size != 0 ) {
       for ( ListIndex = 0; ListIndex < Size; ListIndex++ ) {
             // display_Oid(ListHead[ ListIndex ]);
             if ( !Test_Oid( ListHead[ ListIndex ] , *POid , T_LT ) ) Offset = ListIndex
             if ( Test_Oid( ListHead[ ListIndex ] , *POid , T_EQ ) ) {
                  *PStatus = 0;
                  return(0);
              }
        }


     Offset = Offset * S_OID + S_TypeId ;


     err    = sm_InsertInObject(BufGrp,&SetOid,Offset,S_OID,(void *)POid);
     if (err != esmNOERROR ) ErrorCheck(err,"Insert in Object \n");
     Size   = Size + 1;
     return(0);
   } else {
     err    = sm_AppendToObject(BufGrp,&SetOid,S_OID,(void *)POid);
     if (err != esmNOERROR ) ErrorCheck(err,"Insert in Object \n");
     err    = sm_ReadObject(BufGrp,&SetOid,0,READ_ALL,&UserDesc);
     if (err != esmNOERROR ) ErrorCheck(err,"Insert in Object \n");
     ListHead  = (OID *) (UserDesc->basePtr + S_TypeId);
     Buffer    = UserDesc->basePtr ;
     unpin();
     Size   = Size + 1;
     return(0);
   }
}

ERTYPE Set::deleteFromSet(OID POid,BOOLEAN *PStatus) {
int ListIndex, Offset ;
int err;
OID *CurOid;

     // Find Correct Place To delete Oid .............

     unpin();

     Offset = -1; // set to -1 to check if the element already exists in the set....

     for ( ListIndex=0; ListIndex <= Size; ListIndex++ ) {
          if ( Test_Oid( ListHead[ListIndex] , POid , T_EQ ) ) Offset=ListIndex ;
     }

     if(Offset == -1) {  // element not in the set .....
        *PStatus = 1;
        return(0);       // Escape from routine.........
     } ;

     Offset = Offset * S_OID + S_TypeId ; // Offset#of elements + TYPE of set.....
     err    = sm_DeleteFromObject(BufGrp,&SetOid,Offset,S_OID);
     Size   = Size - 1 ;

     ErrorCheck(err,"Delete From Object \n");

     // Delete below line please....

     *PStatus = 0;
     return(0);
}
```

```
ERTYPE Set::isIn(OID POid,BOOLEAN *PStatus) {
int ListIndex, Offset ;

     // Find Object in set.............

     display_Oid(POid);
     *PStatus = 1 ;      // For the worst case !! .......

     for ( ListIndex=0; ListIndex < Size; ListIndex++ )
           if ( Test_Oid( ListHead[ListIndex] , POid , T_EQ ) ) *PStatus=0;

     // Anything to ask ?...

     return(0);
}

ERTYPE Set::getElement(OID *POid , int SetIndex ,BOOLEAN *PStatus) {

     if ( SetIndex > Size ) {  // If wanted more than set  has ......
          *PStatus = 1 ;
          return (0) ;
     }

     memcpy(POid, &ListHead[ SetIndex ],sizeof(OID));
     *PStatus = 0 ;              // Everything Seems ok...............
     return(0);

}

ERTYPE Set::setUnion(Set PSet,Set USet,BOOLEAN *PStatus  {

int Setdex;
OID Element;

   if ( USet.Size ) {                      // Union Set is not empty .............
       *PStatus = 1 ;
        return(0);
   }

   /*
      Insert first the elements of current set
                                               */

   for (Setdex = 0 ; Setdex <  Size ; Setdex++ )  {


      getElement( &Element, Setdex, PStatus ) ;

      if ( !( *PStatus ) ) {                      // If element received from current
          display_Oid(Element);
          USet.insertToSet( &Element , PStatus );

      }
      else {                                    // If not received ........
          *PStatus = 1;
      }


   }

   /*
      Now Insert Other set's elements ...
```

```
    for (Setdex = 0 ; Setdex < PSet.Size ; Setdex++ )  {

         PSet.getElement( &Element, Setdex, PStatus)  ;

         if ( !( *PStatus ) ) {                           // If element received from second  se

             USet.insertToSet( &Element , PStatus ) ;

         }
         else {
             *PStatus = 1 ;
         }
    }

return( 0 );

}

ERTYPE Set::isSubset(Set PSet,BOOLEAN *PStatus) {
int  SetIndex;
OID  Element;

     SetIndex=0;

     if ( Size > PSet.Size )  {          // If subset, how can it be larger !...
        *PStatus = 1;
        return(0);
     }
                                         // Check all elements in current set if
    do {                                 // They are in the target..........
        getElement(&Element,SetIndex++,PStatus);
        PSet.isIn(Element,PStatus);
    } while ( ( SetIndex < Size)  && ( ! (*PStatus)  ) );

    return(0);
}

ERTYPE Set::isContain(Set PSet, BOOLEAN *PStatus) {
int SetIndex ;
OID Element ;


    SetIndex = 0;

    if ( Size <= PSet.Size ) {       // If PSet is larger no need to check...
       *PStatus =  1;
       return(0);                    // Not an ESM error.....................
    }

    PSet.getFirst(&Element,PStatus);

    if ((*PStatus)) {                // If No elements in current set as can be seen !..
       return(0);                    // Not an ESM error....................
    }

    isIn(Element,PStatus);

    for (SetIndex = 1; SetIndex < PSet.Size ; SetIndex++) {

        PSet.getNext(&Element,PStatus);

        isIn(Element,PStatus);
```

```
              if (*PStatus) return(0);

         }

return(0);                              // No way to get an ESM error !!!...

}

ERTYPE ErrorCheck(int e,char *routine) {

   if(e!=esmNOERROR) {                             // If there is not error do nothing..
     printf("Got a ESM error from %s \n",routine);
     printf("error= %s \n",sm_Error(sm_errno));
     exit(1);                                      // No need to to do this, but for test.
   }
   return(0);
}


OID Set::getSetOid(BOOLEAN *PStatus) {

     if ( OID_IS_INVALID(SetOid) )  // Test if program cheats......
        *PStatus = 1 ;
     else
        *PStatus = 0;

     return(SetOid);                    // May be changed to our standard.!
}

ERTYPE Set::getFirst(OID *POid,BOOLEAN *PStatus) {

        if (ListHead==NULL) {       // There is no set ! or at least not in memory....
           *PStatus = 1;
           POid=NULL;
           return(1);
        }
        memcpy(POid,ListHead,sizeof(OID));// Original point, elements start...(see set def
        *PStatus = 0;
        return(0);


}

ERTYPE Set::getLast(OID *POid,BOOLEAN *PStatus) {

        memcpy(POid , &ListHead[Size],sizeof(OID));
        *PStatus  = 0;            // Think why need  this ?....
        return(0);                // Tell me if you find out !!...

}

void Set::intersect(Set PSet,Set ISet,BOOLEAN *PStatus) {

int SetIndex;
OID Element;

        if ( Size <= PSet.Size ) {
              for ( SetIndex = 0 ; SetIndex < Size ; SetIndex ++ ) {
                     getElement ( &Element , SetIndex , PStatus ) ;
                     PSet.isIn  ( Element  , PStatus );
                     if ( ! (*PStatus ) )
                          ISet.insertToSet ( &Element , PStatus );
              }
              *PStatus = 0;
        }
        else {
            for ( SetIndex = 0 ; SetIndex < PSet.Size ; SetIndex ++ ) {
                PSet.getElement ( &Element , SetIndex , PStatus ) ;
                isIn  ( Element  , PStatus );
                if ( ! (*PStatus ) )
                    ISet.insertToSet ( &Element , PStatus );
            }
            *PStatus = 0;
        }
}
```

```
#include <sequence.h>

void Sequence::setIncr(int Pincr, CatBOOLEAN *PStatus) {
int err;

     if ( ( Pincr ) <  ( ( maxval - minval ) * INCR_MULTIPLIER ) ) {

          err = sm_ReadObject(BufferGroup, &SeqId, 0, READ_ALL,&UserDesc);
          ErrorCheck(err,"increment read");

          increment = Pincr ;
          SeqIntern.increment = Pincr ;

          err = sm_WriteObject(BufferGroup, 0, sizeof(SeqTemp), &SeqIntern,
                         UserDesc,TRUE);
          ErrorCheck(err,"increment write");
                    if ( UserDesc != NULL )
                    err=sm_ReleaseObject(UserDesc);
                    ErrorCheck(err, "releasing sequence object");
     }
     else
          *PStatus = 1 ;

};

void Sequence::setInitial(int Pinit) {

int err;

     err = sm_ReadObject(BufferGroup, &SeqId, 0, READ_ALL, &UserDesc) ;
     ErrorCheck(err,"init read");

     initial = Pinit ;
     SeqIntern.initial = Pinit ;

     err = sm_WriteObject(BufferGroup, 0, sizeof(SeqTemp), &SeqIntern,
                    UserDesc,TRUE);
     ErrorCheck(err,"init write");
     if ( UserDesc != NULL )
          err = sm_ReleaseObject(UserDesc);
          ErrorCheck( err, "release obj. of set initial ");


};

void Sequence::setCycOff() {
int err;


     err = sm_ReadObject(BufferGroup, &SeqId, 0, READ_ALL,&UserDesc);
     ErrorCheck(err,"cyclic read");

     cyclic = 'N' ;
     SeqIntern.cyclic = 'N' ;

     err = sm_WriteObject(BufferGroup, 0, sizeof(SeqTemp), &SeqIntern,
                    UserDesc,TRUE);
     ErrorCheck(err,"cyclic write");
          if ( UserDesc != NULL )
          err = sm_ReleaseObject( UserDesc );
          ErrorCheck(err , "release obj. in cyc off ");

};
```

```
void Sequence::setCycOn() {
int err;

     err = sm_ReadObject(BufferGroup, &SeqId, 0, READ_ALL,&UserDesc);
     ErrorCheck(err,"cyclic on read");

     cyclic = 'Y' ;
     SeqIntern.cyclic = 'Y' ;

     err = sm_WriteObject(BufferGroup, 0, sizeof(SeqTemp), &SeqIntern,
                    UserDesc,TRUE);
     ErrorCheck(err,"cyclic on write");
          if ( UserDesc != NULL )
          err = sm_ReleaseObject( UserDesc );
          ErrorCheck( err, " release obj of cycle on");
};

void Sequence::setMinVal(int Pminval, CatBOOLEAN *PStatus) {
int err;

     if ( Pminval < maxval ) {

          err = sm_ReadObject(BufferGroup, &SeqId, 0, READ_ALL,&UserDesc);
          ErrorCheck(err,"minval read");

          minval = Pminval ;
          SeqIntern.minval = Pminval ;

          if ( minval >= curval ) {
               curval = minval ;
               SeqIntern.minval = minval ;
          } ;

          err = sm_WriteObject(BufferGroup, 0, sizeof(SeqTemp), &SeqIntern,
                         UserDesc,TRUE);
          ErrorCheck(err,"minval write");
                    if ( UserDesc != NULL )
                    err = sm_ReleaseObject( UserDesc ) ;
                    ErrorCheck( err ," release obj. in min val");

     } else
          *PStatus = 1 ;
};

void Sequence::setMaxVal(int Pmaxval, CatBOOLEAN *PStatus) {
int err;

     if ( Pmaxval > minval ) {

          err = sm_ReadObject(BufferGroup, &SeqId, 0, READ_ALL,&UserDesc);
          ErrorCheck(err,"maxval read");

          maxval = Pmaxval ;
          SeqIntern.maxval = Pmaxval ;

          if ( maxval < curval )
               if ( cyclic == 'Y' ) {
                    curval = minval ;
                    SeqIntern.curval = minval ;
               }
               else {
                    curval = maxval ;
                    SeqIntern.curval = maxval ;
```

```
            };

        err = sm_WriteObject(BufferGroup, 0, sizeof(SeqTemp), &SeqIntern,
                                 UserDesc,TRUE);
        ErrorCheck(err,"maxval write");
                      if ( UserDesc != NULL )
                      err = sm_ReleaseObject( UserDesc );
                      ErrorCheck( err, "release obj. in max val of sequence " );

    } else
        *PStatus = 1 ;
};

int  Sequence::getCurVal() {

int err;
    err = sm_ReadObject(BufferGroup, &SeqId, 0, -1 ,&UserDesc);
    ErrorCheck(err,"nextval  read");

    memcpy(&SeqIntern, UserDesc->basePtr, sizeof(SeqTemp));
    curval = SeqIntern.curval ;
        if ( UserDesc != NULL )
        err = sm_ReleaseObject( UserDesc );
        ErrorCheck( err, "release obj. in get cur val of sequence" );
    return(curval);
};

int  Sequence::getNextVal(CatBOOLEAN *PStatus) {

int err;


    err = sm_ReadObject(BufferGroup, &SeqId, 0, -1 ,&UserDesc);
    ErrorCheck(err,"nextval  read");

    memcpy(&SeqIntern, UserDesc->basePtr, sizeof(SeqTemp));

    if ( (SeqIntern.curval + SeqIntern.increment) <=  (SeqIntern.maxval) ) {
          SeqIntern.curval = SeqIntern.curval + SeqIntern.increment ;
          curval = SeqIntern.curval ;

          err = sm_WriteObject(BufferGroup, 0, sizeof(SeqTemp), &SeqIntern,
                                  UserDesc,TRUE);
          ErrorCheck(err,"nextval write");

          *PStatus = 0 ;
          return(curval) ;
    }
    else
          if ( SeqIntern.cyclic == 'Y' )  {
              SeqIntern.curval = SeqIntern.minval ;
              *PStatus = 0 ;
          }
          else
              *PStatus = 1 ;

    curval = SeqIntern.curval ;
    err = sm_WriteObject(BufferGroup, 0, sizeof(SeqTemp), &SeqIntern,
                             UserDesc,TRUE);
    ErrorCheck(err,"nextval write");
        if ( UserDesc != NULL )
        err = sm_ReleaseObject( UserDesc );
        ErrorCheck( err, "nextval release object");
```

```
     return(curval) ;

};

void Sequence::delSeq(CatBOOLEAN *status) {
int err;

     err = sm_DestroyObject(BufferGroup, &SeqId );
     ErrorCheck( err, "delete sequence");
     if ( err != esmNOERROR ) *status = 1 ;
     else { *status = 0;
            cout << "sequence destroyed successfully.n" ;
     };

};

Sequence::Sequence(char *Pseqname, CatBOOLEAN *Pstatus) {

int       DataSize;
FID       SeqFile ;
int       err;
OID       SeqOid;
short     SearchFlag = 0;
CatBOOLEAN  eof;

          //   search for the sequence table...

          DataSize = sizeof(FID);
          err = sm_GetRootEntry(VolumeId, SEQFILE, &SeqFile, &DataSize );
          if ( err == esmNOERROR ) { *Pstatus = 0 ;
               //   search for the required sequence ...
               err = sm_GetFirstOid BufferGroup,&SeqFile,
                              &SeqOid,&ObjectHeader,&eof);
               ErrorCheck(err,"cant get oid");

               if (!eof){ // if file is not empty ........
                   do  { // search if there is a sequence with given name !!!

                       err = sm_ReadObject(BufferGroup, &SeqOid,
                                          0 , -1 , &UserDesc);
                       ErrorCheck(err,"seq.cons.read.obj");
                       memcpy(&SeqIntern,UserDesc->basePtr,sizeof(SeqTemp));

                       if ( !strcmp(SeqIntern.seqname,Pseqname) ) {

                            strcpy(SeqIntern.seqname,Pseqname);

                            memcpy( &SeqId, &SeqOid, sizeof(FID) ) ;

                            initial   = SeqIntern.initial   ;
                            minval    = SeqIntern.minval    ;
                            maxval    = SeqIntern.maxval    ;
                            cyclic    = SeqIntern.cyclic    ;
                            increment = SeqIntern.increment ;
                            curval    = SeqIntern.curval    ;
                        }

                        if ( UserDesc != NULL ) err = sm_ReleaseObject(UserDesc);
                        ErrorCheck(err,"release object");
                        err = sm_GetNextOid(BufferGroup, &SeqOid, &SeqOid,
                                        &ObjectHeader, &eof);
                        ErrorCheck(err,"next object");
                   } while (eof == 0); // check from esm document.....
               };
```

```
        } else *Pstatus = 1;

};

// Create a sequence with the given parameters .....

Sequence::Sequence(char *Pseqname, int Pinit, int Pmin, int Pmax,
                   char Pcyc, int Pinc,CatBOOLEAN *PStatus) {

int       DataSize;
FID       SeqFile ;
OID       SeqOid ;
int       err;
short     SearchFlag = 0;
CatBOOLEAN     eof;


          if ( strchr("YyNn",Pcyc) == NULL ) Pcyc = 'N' ;

          //  search for the required sequence in the sequence file .....
          DataSize = sizeof(FID) ;
          err = sm_GetRootEntry(VolumeId, SEQFILE, &SeqFile, &DataSize);

          if ( err != esmNOERROR ) {
               // There is no sequence file created , so create a new one...
               cout << "Creating Sequence Database....\n";
               err = sm_CreateFile(BufferGroup, VolumeId, &SeqFile );

               ErrorCheck(err,"Seq.cons.create.file");
               err = sm_SetRootEntry(VolumeId, SEQFILE, (char *)&SeqFile, sizeof FID );
               ErrorCheck(err,"Seq.cons.set.root");
               SearchFlag = 1 ; // No need for any search yet !!!!...
          }

          if ( !SearchFlag ) {     // search sequence in file ......
               cout << "searching sequence \n" ;
               err = sm_GetFirstOid(BufferGroup,&SeqFile,&SeqId,&ObjectHeader,
                                    &eof);
               ErrorCheck(err,"Seq.cons.first.oid");

               SearchFlag = 1 ; // if not found in file create sequence ..

               if (!eof)  { // if file is not empty ........
                   do   { // search if there is a sequence with given name !!!

                         err = sm_ReadObject(BufferGroup, &SeqId, 0 , -1 ,
                                             &UserDesc);
                         ErrorCheck(err,"seq.cons.read.obj");

                         memcpy(&SeqIntern,UserDesc->basePtr,sizeof(SeqTemp));

                         if ( !strcmp(SeqIntern.seqname,Pseqname) ) {
                              *PStatus = 1 ;    //is a sequence same  name....
                              SearchFlag = 0 ;
                              eof = 1 ; // exit from loop ....
                         }

                         if ( UserDesc != NULL ) err = sm_ReleaseObject(UserDesc);
                         ErrorCheck(err,"release object");
                         err = sm_GetNextOid(BufferGroup, &SeqId, &SeqOid,
                                             &ObjectHeader, &eof);
                         ErrorCheck(err,"next object");
                         memcpy(&SeqId, &SeqOid, sizeof(OID));
                   } while (eof == 0); // check from esm document.....
               } ;
          };

          if ( SearchFlag ) { // OK. create sequence....
               strcpy(SeqIntern.seqname,Pseqname);
               SeqIntern.initial   = Pinit;
               SeqIntern.minval    = Pmin ;
               SeqIntern.maxval    = Pmax ;
               SeqIntern.cyclic    = Pcyc ;
               SeqIntern.increment = Pinc ;
               SeqIntern.curval    = Pinit;

               err = sm_CreateObject(BufferGroup, &SeqFile,
                                     NEAR_FIRST_PHYSICAL,NULL, NULL ,
                                     sizeof(SeqTemp),&SeqIntern , &SeqOid);

               memcpy(&SeqId,&SeqOid,S_OID);
               ErrorCheck(err,"create sequence");
          };
};
```

```
#include <attribute.h>
extern OID globOid;
extern int Debug;


void CatATTRIBUTE::writeAttr(AttributeType PAttribute, OID *PAttrOid) {
int err;

     err = sm_CreateObject( BufferGroup, &AttributeFid, NEAR_FIRST_PHYSICAL,
                            NULL, NULL , sizeof( AttributeType ), &PAttribute,
                            PAttrOid );
   . ErrorCheck(err, "error writing attribute entry ") ;
/*
     strcpy( (char *)AttrName, (char *)PAttribute.AttributeName);
     memcpy( (char *)&AttrType, (char *)&PAttribute.AttributeType,sizeof(OID) );
     AttrOffset = PAttribute.AttributeOffset;
     AttrLength = PAttribute.AttributeLength;
     Constructor = PAttribute.AttributeConstructor;
     memcpy((char *)&AttrOid , (char *)PAttrOid, sizeof(OID) );
*/
};

void CatATTRIBUTE::disp_attr(AttributeType Pattr) {
     cout << "++++++++++++++++++++++++++++++++++++++++++\n";
     cout << "NAME:      " << Pattr.AttributeName << "\n";
     cout << "OFFSET:    " << Pattr.AttributeOffset << "\n";
     cout << "LENGTH:    " <<  Pattr.AttributeLength << "\n";
     cout << "CONSTRUCT:" << Pattr.AttributeConstructor << "\n";
     cout << "TYPE :" ; display_Oid(Pattr.AttributeType) ;
     cout << "++++++++++++++++++++++++++++++++++++++++++\n";
};

CatATTRIBUTE::CatATTRIBUTE( FID AttrFile ) {

     // initialize the file name to store attributes ...

     memcpy( (char *)&AttributeFid, (char *)&AttrFile, sizeof(FID) );
};

CatATTRIBUTE::CatATTRIBUTE( FID AttrFile , OID *PAttrOid ) {
int err;
AttributeType AttrIntern;

    err = sm_ReadObject(  BufferGroup, PAttrOid , 0 , READ_ALL , &UserDesc );
    ErrorCheck(err, "can not read attribute entry");

    memcpy((char *)&AttrIntern , (char *)UserDesc->basePtr, sizeof(AttrIntern) );
    if ( UserDesc != NULL ) err = sm_ReleaseObject( UserDesc );
    ErrorCheck(err, "release object attribute");

/*
    // initialize member values....
    memcpy( (char *)&AttributeFid, (char *)&AttrFile, sizeof(FID) );
    memcpy( (char *)&AttrOid, (char *) PAttrOid , sizeof(OID) );
    strcpy( (char *)AttrName, (char *)AttrIntern.AttributeName);
    memcpy( (char *)&AttrType, (char *)&AttrIntern.AttributeType, sizeof(OID) );
    AttrOffset = AttrIntern.AttributeOffset ;
    AttrLength = AttrIntern.AttributeLength ;
    Constructor = AttrIntern.AttributeConstructor ;
*/

};
```

```
void CatATTRIBUTE::readAttr(OID AttributeOid,AttributeType *AttrIntern, CatBOOLEAN *statu
int err ;
       err = sm_ReadObject(BufferGroup, &AttributeOid, 0 , READ_ALL,&UserDesc);
       ErrorCheck(err, "error on reading attribute entry");
       if ( err != esmNOERROR) *status = 1 ;
       else { // succeded to read attribute ...
          memcpy(( char * )AttrIntern,( char *)UserDesc->basePtr, sizeof( AttributeType ) ;
          // initialize member values....
          *status = 0 ;
          if ( UserDesc != NULL ) err = sm_ReleaseObject(UserDesc);
          ErrorCheck(err, "error on release");
     };

};
void CatATTRIBUTE::updateAttr(OID Pattroid, AttributeType Pattrinfo ) {
int err;

          err = sm_ReadObject( BufferGroup, &Pattroid, 0, READ_ALL, &UserDesc );
          ErrorCheck( err, "reading updated object");

          err = sm_WriteObject( BufferGroup, 0, sizeof(AttributeType), &Pattrinfo ,
                                                                    UserDesc, TRUE);
     ErrorCheck( err, "writing updated object");
          if ( UserDesc != NULL ) sm_ReleaseObject( UserDesc );

};
```

```
#include <catalog.h>
#include <strings.h>
extern int User;
#define check( a ,b ) { if (*a) { cout << b << "\n" ; return; }; } ;

void MoodsTypes::dispLevel( LevelInfo * Plevel ) {

            if ( !Plevel ) return;

            cout << Plevel->ClassName << "," ;
            dispLevel( Plevel->next );
            cout << "\n" ;

};

void MoodsTypes::dispHier( Hierarchy *Phier ) {

      if ( !Phier ) return;

      dispLevel( Phier->thislevel );
            cout << "-----------------\n" ;
            dispHier( Phier->nextlevel );
};

void MoodsTypes::changeNameOfClass( char *Pold, char *Pnew ) {
CatBOOLEAN status;
Catalog tmpType;
AttributeType tmpAttr;
OID     tmpOid,curOid,nextOid;
CatBOOLEAN eof;
int      err;

           getTypOid( Pold, &tmpOid, &status );
           check( &status, "can not find type to change name" );

           readType( tmpOid , &tmpType, &status );
           check( &status, "can not read type information ");

           strcpy(tmpType.TypeName , Pnew );
           updateType( tmpOid, tmpType );

           // search all attributes to change AttributeTypeName field...
                             CatATTRIBUTE attrins(AttributeId);

           err = sm_GetFirstOid( BufferGroup, &AttributeId, &curOid, &ObjectHeader, &eof );
           if ( err != esmNOERROR ) { status = 1 ; return; } ;
           if ( !eof ) {
                 do {
                       err = sm_ReadObject(BufferGroup, &curOid, 0, -1, &UserDesc );
                       if ( err != esmNOERROR ) { status = 1 ; return; } ;
                       memcpy( &tmpAttr, UserDesc->basePtr, sizeof(AttributeType) );
                       if ( !strcmp( tmpAttr.AttrTypeName, Pold ) ) {
                             strcpy( tmpAttr.AttrTypeName , Pnew );
                             attrins.updateAttr( curOid, tmpAttr );
                       };
                       if ( UserDesc != NULL ) {
                             err = sm_ReleaseObject( UserDesc ) ;
                             ErrorCheck( err, "can not release attribute object" );
                       };
                                                       err = sm_GetNextOid( BufferGroup,&curOid
                                                       memcpy( &curOid, &nextOid, S_OID );
                 } while ( !eof ) ;
     };
```

```
};

void MoodsTypes::addAttributeToClass(char *PClaName, AttributeType PAttr ,CatBOOLEAN *sta
{
Catalog tmpType;
AttributeType tmpAttr;
OID     tmpOid,AttrOid;
int     l_type,l_size,err;


        getTypOid( PClaName, &tmpOid, status );
        check( status, "can not find class to add attribute" );

        readType( tmpOid, &tmpType, status );
        check( status , "can not read type information" );

        if ( !memcmp( &tmpType.List_SubClass , &nullOid , S_OID ) ) {
              cout << " this class has some sub classes. not implemented yet \n" ;
              *status = 1 ;
              return;
        };
        List subClsList( &tmpType.List_SubClass , &l_type, &l_size, status ) ;
        if ( ! (*status) )
              if ( l_size ) {
                    cout << "this class has some sub classes. not implemented yet \n" ;
                    return;
              };
        // i should write this!!!!!!!!
        CatATTRIBUTE myattr(AttributeId) ;

        // delete instance file & recreate it from scratch...
        err = sm_DestroyFile( BufferGroup , &tmpType.Instance_File );
        ErrorCheck( err , "can not destroy instance file of class" );

        err = sm_CreateFile( BufferGroup, VolumeId, &tmpType.Instance_File);
        ErrorCheck( err, "can not create new instance file" );

        List AttrList( &tmpType.List_Attributes, &l_type, &l_size, status );
        check( status, "can not read attribute list");

        myattr.writeAttr(PAttr, &AttrOid );

        AttrList.insertToList( &AttrOid, status );
        check( status, "can not insert to attribute list" );

        updateType( tmpOid, tmpType ); // just for instance file..
};



void MoodsTypes::getClassHierarchy(char *PClaName, Hierarchy **PHierarchy,CatBOOLEAN *sta
{
Catalog Intern,tmpType;
OID     tmpOid,element;
Hierarchy *top_level;
LevelInfo *nowhereLevel;


int     indx,l_type,l_size;

        // the structure will be filled dynamically.. destroy them...

        getTypOid( PClaName, &tmpOid, status) ;
        check( status, "can not find requested class" );
        //  read type info to get sub class list into memory ....
        readType( tmpOid, &Intern, status );
        check( status, "can not read type info");
        List subcls( &Intern.List_SubClass, &l_type, &l_size , status ) ;
        check( status, "can not construct sub class list");
        if ( l_size ) { // pre allocation of hierarchy....
             *PHierarchy = new Hierarchy ;
             top_level = *PHierarchy;
             top_level->thislevel = NULL ;
             top_level->nextlevel = NULL ;

             top_level->thislevel = new LevelInfo ;
             nowhereLevel = top_level->thislevel ;
             nowhereLevel->next = NULL ;

             // put father into hierarchy ...
             strcpy( nowhereLevel->ClassName, Intern.TypeName ) ;
        };

        // add all subclasses to the end of list...

        for ( indx = 0 ; indx < l_size ; indx++ ) {
             subcls.getElement( &element, indx, status ) ;
             check( status, "can not receive element from list" );

             readType( element, &tmpType, status ) ;
             check( status, "can not read type info");

             nowhereLevel->next = new LevelInfo ;
             nowhereLevel = nowhereLevel->next  ;
             nowhereLevel->next = NULL ;

             strcpy( nowhereLevel->ClassName , tmpType.TypeName );
        };

        for ( indx = 0 ; indx < l_size ; indx ++ ) {
             subcls.getElement( &element, indx, status );
             check( status ,"can not receive element from list") ;

             readType( element, &tmpType, status ) ;
             check( status, "can not read type info") ;
             getClassHierarchy( tmpType.TypeName, &top_level->nextlevel, status);
                                  if ( top_level->nextlevel )  top_level = top_level
             check( status,"error in inner construct of hierarchy") ;

        };

};
void MoodsTypes::dispSuperClass( char *Pcls ) {

CatBOOLEAN status;
OID typid, listid;
int indx,l_size,l_type;

Catalog typeentry,tmpentry;

        getTypOid( Pcls, &typid, &status ) ;
        check (&status,"can not find class in disp superclass" );

        readType( typid, &typeentry, &status);
        check( &status, "can not read type entry");

        List SuperList( &typeentry.List_SuperClass, &l_type, &l_size, &status);
```

```
            check ( &status, "can not construct list superclass" );

            for ( indx=0; indx < l_size; indx++ ) {

                        SuperList.getElement( &listid, indx, &status);
                        check( &status, "can not get element from list");

                        readType( listid, &tmpentry, &status );
                        check( &status, "can not read type info ");

                        cout << tmpentry.TypeName << "    .n" ;
            };
};

void MoodsTypes::dispSubClass( char *Pcls ) {

CatBOOLEAN status;
OID typid, listid;
int indx,l_size,l_type;

Catalog typeentry,tmpentry;

            getTypOid( Pcls, &typid, &status ) ;
            check (&status,"can not find class in disp superclass" );

            readType( typid, &typeentry, &status);
            check( &status, "can not read type entry");

            List SuperList( &typeentry.List_SubClass, &l_type, &l_size, &status);
            check ( &status, "can not construct list superclass" );

            for ( indx=0; indx < l_size; indx++ ) {

                        SuperList.getElement( &listid, indx, &status);
                        check( &status, "can not get element from list");

                        readType( listid, &tmpentry, &status );
                        check( &status, "can not read type info ");

                        cout << tmpentry.TypeName << "    .n" ;
            };
};

void MoodsTypes::getSubClass( char *Pcls , ClassNames **Pname) {

CatBOOLEAN status;
OID typid, listid;
int indx,l_size,l_type;

Catalog typeentry,tmpentry;
ClassNames *nowhere;

            getTypOid( Pcls, &typid, &status ) ;
            check (&status,"can not find class in disp superclass" );

            readType( typid, &typeentry, &status);
            check( &status, "can not read type entry");

            List SuperList( &typeentry.List_SubClass, &l_type, &l_size, &status);
            check ( &status, "can not construct list superclass" );

            if ( l_size ) {
                        *Pname = new ClassNames ;
                        nowhere = *Pname ;
```

```
                    nowhere->next = NULL ;
    };

        for ( indx=0; indx < l_size; indx++ ) {

            if ( !indx ) {
                    *Pname = new ClassNames ;
                        nowhere = *Pname ;
                        nowhere->next = NULL ;
            } else {
                            nowhere->next = new ClassNames ;
                            nowhere = nowhere->next ;
                        nowhere->next = NULL ;
            };

                    SuperList.getElement( &listid, indx, &status);
                    check( &status, "can not get element from list");

                    readType( listid, &tmpentry, &status );
                    check( &status, "can not read type info ");
                    strcpy( nowhere->Name , tmpentry.TypeName );

    };
};

void MoodsTypes::getSuperClass( char *Pcls , ClassNames **Pname) {

CatBOOLEAN status;
OID typid, listid;
int indx,l_size,l_type;

Catalog typeentry,tmpentry;
ClassNames *nowhere;

        getTypOid( Pcls, &typid, &status ) ;
        check (&status,"can not find class in disp superclass" );

        readType( typid, &typeentry, &status);
        check( &status, "can not read type entry");

        List SuperList( &typeentry.List_SuperClass, &l_type, &l_size, &status);
        check ( &status, "can not construct list superclass" );

        if ( l_size ) {
                    *Pname = new ClassNames ;
                    nowhere = *Pname ;
                    nowhere->next = NULL ;
    };

        for ( indx=0; indx < l_size; indx++ ) {

            if ( !indx ) {
                    *Pname = new ClassNames ;
                        nowhere = *Pname ;
                        nowhere->next = NULL ;
            } else {
                            nowhere->next = new ClassNames ;
                            nowhere = nowhere->next ;
                        nowhere->next = NULL ;
            };

                    SuperList.getElement( &listid, indx, &status);
                    check( &status, "can not get element from list");
```

```
                    readType( listid, &tmpentry, &status );
                    check( &status, "can not read type info ");
                    strcpy( nowhere->Name , tmpentry.TypeName );

    };
};

void MoodsTypes::getClassInfo(char *PClsName,Catalog *PCat,AttrList **PAttr,
                    ClassNames **Psuper, ClassNames **Psub, CatBOOLEAN *status ) {
OID     typeid,tmpoid;
Catalog intern,tmpinfo;
int     l_type, l_size,indx;
ClassNames *nowhere,*top;

        getSuperClass(PClsName,Psuper);
        getSubClass(PClsName,Psub);
        getClassInfo( PClsName, PCat, PAttr, status );
};

void MoodsTypes::getClassInfo(char *PClsName,Catalog *PCat,AttrList **PAttr,CatBOOLEAN *s
OID         typeid;
Catalog     intern;

            getTypOid( PClsName, &typeid, status );
            check( status, "can not find type info" );

            readType( typeid, &intern, status );
            check( status, "can not read type info" );

            memcpy( PCat , &intern, sizeof(Catalog) );

            getAttrInfo( typeid , PAttr, status );
            check( status , "can not construct attribute " );

};


void MoodsTypes::getAttrInfo(OID PType ,AttrList **PAttr,CatBOOLEAN *status)
{
OID typeid, attrid;
Catalog     intern;
int         indx,l_type,l_size;
char y[1000];
AttributeType intern_attr;
char z[1000];
AttrList    attr_list;
AttrList    *list_start, *nowhere=NULL;


            readType( PType , &intern, status );
            check( status, "can not read type info" );

            List a_list( &intern.List_Attributes, &l_type, &l_size, status );
            check( status, " can not construct attr list" );

            CatATTRIBUTE attr_instance(AttributeId);
            for ( indx = 0 ; indx < l_size ; indx++ ) {
                a_list.getElement( &attrid, indx, status );
                check( status, "can not get entry from list" );

                if (nowhere != NULL ) {
                    nowhere->next = new AttrList;
                    nowhere = nowhere->next ;
                } else
```

```
                                                          nowhere = new AttrList ;

                nowhere->next = NULL ;
                nowhere->nextsub = NULL ;

                if (!indx) list_start = nowhere ;

                attr_instance.readAttr( attrid , &intern_attr, status ) ;
                nowhere->entry = intern_attr ;
                if ( intern_attr.AttributeConstructor != 'N' ) {
                     getAttrInfo(intern_attr.AttributeType,&nowhere->nextsub,status );
                     check( status , "can not get internal attribute information");
                } else
                                                           nowhere->nextsub = NULL ;

          };
          *PAttr = list_start ;
          *status = 0 ;
};


void HoodsTypes::printvals(Catalog toPrint) {

cout << "----------------------------------------\n" ;
cout << "type name=" << toPrint.TypeName << "\n" ;
cout << "type   id=" << toPrint.TypeId   << "\n" ;
cout << "Size     =" << toPrint.Size     << "\n" ;
cout << "Cls. Type=" << toPrint.ClassType << "\n" ;
cout << "----------------------------------------\n" ;
};

void HoodsTypes::dispTypes() {
Catalog intern;
OID     curType;
int     curId=1;
CatBOOLEAN status=0;

        getTypOid(curId++,&curType,&status);
        if (status) {
             cout << "error reading first type\n";
             return;
        };
        while (!status) {
                readType(curType, &intern,&status);
                if ( status ) {
                     cout << "DT:can not read type info\n";
                     return;
                };
                printvals(intern);
                if (status) cout << "error reading type\n";
                getTypOid(curId++,&curType,&status);
                if ( status ) {
                     cout << "DT:can not read type oid\n";
                     return;
                };
        };
};
void HoodsTypes::describe( char *Pclsname ) {
OID tmpOid;
CatBOOLEAN status ;

     getTypOid( Pclsname, &tmpOid, &status );
     check( &status, "type not found");
     describe(tmpOid);
```

```
};


void HoodsTypes::describe( OID PclsName, int left_tab  = 0) {
OID attrOid;
int l_type,l_size,indx;
int par=0, i ;
CatBOOLEAN status=0;
char    sp2[1000];
Catalog catentry;
char    sp[1000];
Catalog catentryo;
char    x[1000];
AttributeType attrinfo;

     readType( PclsName, &catentry, &status) ;
     check( &status, "cant read type entry" );

     if ( !memcmp( &catentry.List_Attributes, &nullOid, S_OID ) )
          .. no attributes for this type...
          return;

     List attrlist( &catentry.List_Attributes, &l_type, &l_size, &status );
     check( &status, "can not construct list in describe" );

     CatATTRIBUTE curattr( AttributeId )   ;

     for ( indx = 0 ; indx < l_size ; indx++ ) {
           attrlist.getElement( &attrOid, indx, &status );
           for  i = 0; i < left_tab; i++ ) cout<<' ';
           check(&status, "can not receive attribute oid" );

           curattr.readAttr( attrOid,&attrinfo, &status );
           check(&status, "can not read attribute entry in describe" );

           cout << attrinfo.AttributeName << " ";

           switch( attrinfo.AttributeConstructor ) {
                   case 'T' : left_tab +=3; cout << "TUPLE(\n "; par++; break;
                   case 'S' : left_tab +=3; cout << "SET(\n "; par++; break;
                   case 'R' : left_tab +=3; cout << "REF(\n "; par++; break;
                   case 'L' : left_tab +=3; cout << "LIST(\n "; par++; break;
           };

           readType( attrinfo.AttributeType, &catentryo, &status);
           check( &status , "cant read type info in describe" );

           cout << catentryo.TypeName << " " ;
           describe( attrinfo.AttributeType,left_tab) ;
             while ( par > 0 ) {
                   par = par - 1 ;
                   for( i = 0; i < left_tab; i++ ) cout << ' ';
                   left_tab -= 3;
                   cout << ")" ;
           };

           while ( par > 0 ) {
                   par = par - 1 ;
                   for( i = 0; i < left_tab; i++ ) cout << ' ';
                   left_tab -= 3;
                   cout << ")" ;
           };
           cout << "\n";
     };
```

```
};



void MoodsTypes::getAttrFile( FID *Pfid ) {
     memcpy( Pfid, &AttributeId, S_FID );
};
void MoodsTypes::deleteFiles() {
int err;
CatBOOLEAN status=0;

     err = sm_DestroyFile( BufferGroup, &CatalogId );
     ErrorCheck( err , "deleting catalog file" );
     err = sm_DestroyFile( BufferGroup, &AttributeId );
     ErrorCheck( err, "deleting attribute file" );
     err =  sm_DestroyFile( BufferGroup, &FunctionId );
     Sequence current(SEQNAME, &status ) ;
     if ( status ) cout << " existing sequence could not be read " ;
     else current.delSeq( &status ) ;
     if ( status ) cout << "stg error on sequence...\n" ;


     Sequence newSeq(SEQNAME, 0, 0, 2000, 'N', 1, &status ) ;
     if ( status ) cout << "sequence could not be created.\n" ;


};


MoodsTypes::MoodsTypes(FID MoodsTypeFile) {

OID      FilesOid;
CatBOOLEAN eof,status;
Internal_Files File_data;
int err ;

    // open catalog given catalog file....
    cout << "trying to get moodstypes.\n";
    memset(&nullOid,0,S_OID);
    memset(&nullFid,0,S_FID);

    memcpy(&CatalogId,&MoodsTypeFile,sizeof(FID));

    err = sm_GetFirstOid(BufferGroup, &CatalogId, &FilesOid,
                         &ObjectHeader, &eof);
    ErrorCheck(err,"cant get first oid");

    if ( (err != esmNOERROR ) || eof ) { // if no attribute & function files create...

       err = sm_CreateFile(BufferGroup, VolumeId, &AttributeId);
       ErrorCheck(err, "creating attribute file");
       sm_CreateFile(BufferGroup, VolumeId, &FunctionId);
       ErrorCheck(err, "creating function file");

       // copy FID values of created files to internal variables.....

       memcpy(&File_data.FileAttr, &AttributeId, sizeof(FID));
       memcpy(&File_data.FileFunc, &FunctionId, sizeof(FID));

       // put FID values of the assoc. files to the beginning of catalog file...

       err = sm_CreateObject(BufferGroup, &CatalogId,NEAR_FIRST_PHYSICAL,NULL,
                              NULL, sizeof(Internal_Files),
                              &File_data, &FilesOid);
       ErrorCheck(err,"error writing header");

    } else {

       // get the FID values of assoc. files to the member variables of class...

       err = sm_ReadObject(BufferGroup, &FilesOid, 0 , -1 , &UserDesc);
       ErrorCheck(err, "error on reading header");

       memcpy(&File_data, UserDesc->basePtr, sizeof(Internal_Files));

       if ( UserDesc != NULL ) err = sm_ReleaseObject( UserDesc );
       ErrorCheck(err, "release object of moodstypes" );

       //  copy related file identifiers to internal variables ....

       memcpy(&AttributeId,&File_data.FileAttr,sizeof(FID));
       memcpy(&FunctionId,&File_data.FileFunc,sizeof(FID));
       getTypOid("MoodsRoot", &rootOid, &status) ;
                check( &status, "no MoodsRoot is defined" );
    } ;
};


void MoodsTypes::deleteAttributes( OID PAttlist, CatBOOLEAN *Pstatus ) {

OID element;
int indx,l_type,l_size, err;
Catalog typeinfo;

      List dellist( &PAttlist, &l_type, &l_size, Pstatus ) ;
      check( Pstatus, "can not construct list " );

      CatATTRIBUTE subattr(AttributeId) ;

      for ( indx= 0 ; indx < l_size ; indx++ ) {
            dellist.getElement( &element, indx, Pstatus ) ;
            check( Pstatus , "error on getting element from attribute list" );

            readType(element, &typeinfo, Pstatus ) ;
            if ( !strcmp (typeinfo.TypeName," ") )
                 deleteAttributes( typeinfo.List_Attributes, Pstatus) ;
            err = sm_DestroyObject( BufferGroup, &element );
            ErrorCheck( err, "can not delete attribute" );
      };
      err = sm_DestroyObject( BufferGroup, &PAttlist) ;
};

void MoodsTypes::deleteType( OID Ptype ) {
int err;

     err = sm_DestroyObject( BufferGroup, &Ptype) ;

};



void MoodsTypes::dropFromSuperClass( OID PClass, OID PList, CatBOOLEAN *Pstatus ) {
int      indx,indxo,l_type,l_size,l_typeT,l_size ;
OID      element,elemento;
Catalog typeinfo;
```

```
        if ( memcmp( &PList , &nullOid, S_OID ) ) { // list not empty ....
              List superlist( &PList, &l_type, &l_size, Pstatus ) ;
              check( Pstatus, "unable to construct list" );

              for ( indx = 0 ; indx < l_size ; indx++ ) {
                     superlist.getElement( &element ,indx, Pstatus ) ;
                     check( Pstatus , "unable to receive element from list" );

                     readType( element, &typeinfo, Pstatus) ;
                     check( Pstatus , "unable to read type information");

                     if ( memcmp( &typeinfo.List_SubClass, &nullOid, S_OID ) ) {
                            // super class has some sub class list ...
                            List subclass(&typeinfo.List_SubClass,&i_type,&i_size,Pstatus);
                            check( Pstatus , " can not construct intern. subclass " );

                            for ( indxo = 0 ; indxo < i_size ; indxo++ ) {
                                   subclass.getElement( &elemento, indxo, Pstatus) ;
                                   check( Pstatus, "cant receive element from list" );

                                   if ( !memcmp( &PClass, &elemento, S_OID ) ) {
                                          // my type is found...
                                          subclass.deleteFromList( indxo, indxo, Pstatus);
                                          check( Pstatus,"error deleting from super cls");

                                   };
                            };
                     } else {
                            *Pstatus = 1 ;
                            check( Pstatus , "super dont have sub as me" );
                     };
              };
        } else {
              *Pstatus = 1 ;
              check( Pstatus, "no elements in super class's sub list" );
        };
};

void MoodsTypes::deleteClass( char *PclsName, int *Pforce, CatBOOLEAN *status ) {
OID tmpOid;
     getTypOid(PclsName, &tmpOid, status );
            check(status, "can not find type entry" );
            deleteClass( tmpOid, Pforce, status);
};

void MoodsTypes::deleteClass( OID PClass, int *PForce, CatBOOLEAN *Pstatus ) {
Catalog delClass;
int      l_type,l_size, indx ;
OID      element ;
int      err;


        readType( PClass, &delClass, Pstatus );
        check( Pstatus, "class non existant" );

        if ( !memcmp( &delClass.List_SubClass, &nullOid, S_OID ) ) {
              // no inherited classes ( LEAF LEVEL ) .....
              dropFromSuperClass( PClass, delClass.List_SuperClass, Pstatus );
              check( Pstatus, "can not delete from super class" );

              // current type is deleted from all super classes
              deleteAttributes( delClass.List_Attributes , Pstatus );
              check( Pstatus, "can not delete attributes of class " );
                                   *// you can not understand this please don't bother..

              err = sm_DestroyFile( BufferGroup, &delClass.Instance_File );
              ErrorCheck( err, "destroying instance file" );

              deleteType( PClass ) ;
              *Pstatus = 0 ;
        } else { // contains sub classes ....
              if ( *PForce ) { // sure delete....
                     List subcls( &delClass.List_SubClass, &l_type, &l_size, Pstatus) ;
                     check( Pstatus, "error constructing sub class list");

                     for ( indx = 0 ; indx < l_size ; indx++ ) {
                            // delete all sub classes ...
                            subcls.getElement( &element, indx, Pstatus );
                            check( Pstatus, "can not get subclass from list" );

                            deleteClass( element, PForce, Pstatus );
                            check( Pstatus, "error on delete class" );
                     };
                                          if ( !l_size ) { // no entries in list....
                                                               dropFromSuperClass( PClass, del
                                                               check( Pstatus , "can not delet
                                                               *
                                                               deleteAttributes( delClass.List
                                                               check( Pstatus, "can not delete
                                                               *     you can not understand th

                                                               err = sm_DestroyFile( BufferGro
                                                               ErrorCheck( err, "destroying in

                                                               deleteType( PClass);
                                                               return;

                     };

                     *PForce = 0 ;
                     *Pstatus = 0 ;
              } else {
                     *PForce = 1 ;
                     *Pstatus = 1 ;
              };
        };
        err = sm_DestroyFile( BufferGroup, &delClass.Instance_File );
        ErrorCheck( err, "destroying instance file" );
};

void MoodsTypes::writeType( Catalog Pcat, OID *Ptypeoid, CatBOOLEAN *Pstatus) {
Catalog tmpCat;
int err;

        Sequence typeIdgen(SEQNAME, Pstatus) ;
        check( Pstatus , "can not create type id generator" );

        memcpy( &tmpCat, &Pcat, S_CATALOG );
        tmpCat.TypeId = typeIdgen.getNextVal( Pstatus ) ;
        check( Pstatus , "can not get next value from type id generator" );

        err = sm_CreateObject(BufferGroup,&CatalogId,NEAR_FIRST_PHYSICAL,NULL,
                              NULL,S_CATALOG,&tmpCat,Ptypeoid);
        ErrorCheck( err, "error writing type information" );
        cout << "writing type...\n" ;
        printvals( tmpCat );
};

int MoodsTypes::findType( char *PClsName ) {

CatBOOLEAN status ;
OID     spare;

        getTypOid( PClsName, &spare, &status );
        if ( !status ) { cout << "type found\n" ;return(1); }
                                  else { cout << "type not found\n" ;return(0); };
};

void MoodsTypes::updateType( OID Ptypeoid, Catalog Ptypeinfo ) {
int     err ;

     err = sm_ReadObject(  BufferGroup, &Ptypeoid,  0, READ_ALL, &UserDesc );
     ErrorCheck( err, "reading type info " );

     err = sm_WriteObject( BufferGroup, 0, S_CATALOG, &Ptypeinfo, UserDesc, TRUE );
     ErrorCheck( err, "update type entry" );
          if ( UserDesc != NULL ) sm_ReleaseObject ( UserDesc ) ;

};

void MoodsTypes::copyInheritedTypes( OID *Pinherit, List *Psuperlist, int *Poffset,
                                     CatBOOLEAN *Pstatus ) {
Catalog typeinfo;
int     l_type,l_size ;
OID     attroid,tmpoid;
AttributeType attrinfo;

     readType( *Pinherit, &typeinfo, Pstatus );
     check( Pstatus, "can not read super class info ");

     List attrlist(&typeinfo.List_Attributes, &l_type, &l_size, Pstatus );
     check ( Pstatus , "can not construct attribute list " );

     CatATTRIBUTE curattr(AttributeId) ;

     attrlist.getFirst(&attroid, Pstatus);
     check( Pstatus, "can not get first attribute" );

     curattr.readAttr( attroid, &attrinfo, Pstatus );
     check( Pstatus, "can not read attribute information" );

     do {
          attrinfo.AttributeOffset = *Poffset ;
          *Poffset = *Poffset + attrinfo.AttributeLength ;
          attrinfo.ClassMember = 0 ;
                         cout << "writing attribute info as...:\n" ;
                         printf("%x\n",attrinfo.ClassMember);
          curattr.writeAttr( attrinfo , &tmpoid );
          Psuperlist->insertToList( &tmpoid, Pstatus );
          check( Pstatus, "error on inserting attribute list" );

          attrlist.getNext(&attroid, Pstatus);

          if ( !(*Pstatus) )
               curattr.readAttr(attroid, &attrinfo, Pstatus) ;

     } while ( !(*Pstatus) ) ;
     *Pstatus = 0 ;
};
```

```
void MoodsTypes::linkToSuper( OID Pinherits, OID Pmytype ) {
CatBOOLEAN status;
int     l_type,l_size,i_size,i_type,indx;
OID     oid,tmpoid,inheritsoid;;
Catalog curtype;

     // construct a list of super classes ....
     List superCls( &Pinherits, &l_type, &l_size, &status );
     check ( &status, "can not construct super class list" );

     for ( indx = 0 ; indx < l_size ; indx++ ) {
          superCls.getElement( &oid, indx, &status );
          check( &status, "can not get super class oid" );

          // read type info of super class...
          readType( oid, &curtype, &status );
          check( &status, "can not read super class type" );

          if ( !memcmp( &curtype.List_SubClass, &nullOid, S_OID ) ) {
               // null list ( not created yet ) ..... (create it)...
               List sublist(&CatalogId,NEAR_FIRST_PHYSICAL,NULL,MAXINHERITS,&status );
               check( &status, "can not construct subclass list" );

               sublist.insertToList(&Pmytype, &status);
               check( &status, "can not insert mytype to subclass list" );

               sublist.getListOid( &tmpoid, &status );
               check( &status, "can not get lists oid" );

               memcpy(&curtype.List_SubClass, &tmpoid, S_OID );

               updateType( oid, curtype );
          } else { // sub class list exists ....

               // load list into memory....
               List sublist( &curtype.List_SubClass,&i_type,&i_size,&status );
               check( &status, "error on getting sub class list ");

               sublist.insertToList( &Pmytype, &status);
               check( &status, "can not insert mytype to subclass list" );

          };
     };
};

void MoodsTypes::getTypOid( char *PClsName, OID *PTypeOid, CatBOOLEAN *Pstatus) {

int err;

OID curOid,nextOid;
CatBOOLEAN eof;
Catalog tinfo;

               if ( PClsName == NULL ) { *Pstatus = 1 ; return ; } ;
               if ( !strcmp( PClsName, " ") ) { *Pstatus = 1; return ; } ;

       // search for the entry with the given name ...
       err = sm_GetFirstOid( BufferGroup, &CatalogId, &curOid, &ObjectHeader, &eof);
       if ( err != esmNOERROR ) { *Pstatus = 1 ; return; } ;
       if ( !eof ) {
            do { // start search...
               err = sm_ReadObject(BufferGroup, &curOid, 0, -1, &UserDesc  ;
               if ( err != esmNOERROR ) { *Pstatus = 1 ; return; } ;
            memcpy( &tinfo, UserDesc->basePtr, S_CATALOG );
            if ( !strcmp( tinfo.TypeName, PClsName ) ) {
                 memcpy( PTypeOid, &curOid, S_OID );
                 *Pstatus = 0;
                 if ( UserDesc != NULL ) err = sm_ReleaseObject(UserDesc);
                 ErrorCheck( err, "release object" );
                 return;
            };
            if ( UserDesc != NULL ) err = sm_ReleaseObject(UserDesc);
            ErrorCheck( err , "release object" );
            err = sm_GetNextOid(BufferGroup,&curOid,&nextOid,&ObjectHeader,&eof);
            if ( err != esmNOERROR ) { *Pstatus = 1 ; return; } ;
            memcpy( &curOid, &nextOid, S_OID );
         } while ( !eof ) ;
      };
      memcpy( PTypeOid, &nullOid, S_OID );
      *Pstatus = 1 ;
};

void HoodsTypes::getTypOid(int typeid , OID *PTypeOid, CatBOOLEAN *Pstatus) {

int err;

OID curOid,nextOid;
CatBOOLEAN eof;
Catalog tinfo;

       // search for the entry with the given name ...
      err = sm_GetFirstOid( BufferGroup, &CatalogId, &curOid, &ObjectHeader, &eof);
      if ( err != esmNOERROR ) { *Pstatus = 1 ; return; } ;
      if ( !eof ) {
         do { // start search...
            err = sm_ReadObject(BufferGroup, &curOid, 0, -1, &UserDesc );
            ErrorCheck( err, "Exodus:internal errore" );
            memcpy( &tinfo, UserDesc->basePtr, S_CATALOG );
            if ( tinfo.TypeId == typeid ) {
                 memcpy( PTypeOid, &curOid, S_OID );
                 *Pstatus = 0;
                 if ( UserDesc != NULL ) err = sm_ReleaseObject(UserDesc);
                 ErrorCheck( err, "release object" );
                 return;
            };
            if ( UserDesc != NULL ) err = sm_ReleaseObject(UserDesc);
            ErrorCheck( err , "release object" );
            err = sm_GetNextOid(BufferGroup,&curOid,&nextOid,&ObjectHeader,&eof);
            if ( err != esmNOERROR ) { *Pstatus = 1 ; return; } ;
            memcpy( &curOid, &nextOid, S_OID );
         } while ( !eof ) ;
      };
      memcpy( PTypeOid, &nullOid, S_OID );
      *Pstatus = 1 ;
};


void HoodsTypes::createInstanceFile( FID * PFid, CatBOOLEAN *Pstatus ) {

int err;

     err = sm_CreateFile( BufferGroup, VolumeId, PFid);
     ErrorCheck( err, " creating instance file" );
          cout << "CREATING INSTANCE FILE FOR CLASS\n" ;
     if ( err != esmNOERROR ) *Pstatus = 1 ;
     else *Pstatus = 0;
```

```
};

void HoodsTypes::readType( OID PtypeOid, Catalog *Pcat, CatBOOLEAN *Pstatus ) {

int err ;

     err = sm_ReadObject( BufferGroup, &PtypeOid, 0, -1, &UserDesc ) ;
     ErrorCheck( err, "reading catalog object" );

     if (err != esmNOERROR) *Pstatus = 1 ;
     else  {
               memcpy( Pcat, UserDesc->basePtr, sizeof(Catalog) ) ;
               if ( UserDesc != NULL ) err = sm_ReleaseObject(UserDesc);
               ErrorCheck( err, "on release object");
               *Pstatus = 0 ;
        } ;
};

void HoodsTypes::readType( char *PClsName, Catalog *Pcat, CatBOOLEAN *Pstatus ) {

int err ;
OID tmpOid;
     getTypOid( PClsName, &tmpOid, Pstatus );
     err = sm_ReadObject( BufferGroup, &tmpOid, 0, -1, &UserDesc ) ;
     ErrorCheck( err, "reading catalog object" );

     if (err != esmNOERROR) *Pstatus = 1 ;
     else  {
               memcpy( Pcat, UserDesc->basePtr, sizeof(Catalog) ) ;
               if ( UserDesc != NULL ) err = sm_ReleaseObject(UserDesc);
               ErrorCheck( err, "release object readtype" );
               *Pstatus = 0 ;
     } ;

};

void HoodsTypes::setClassType( char *Ptype, int User ) {

     if ( User == 0 ) *Ptype = 'C' ;
     else if ( User == 1 ) *Ptype = 'c' ;
     else *Ptype = 't' ;

};

void HoodsTypes::createUnnamedType( memberNode *Pattr, int *offset, OID *Ptypeoid,
                                    CatBOOLEAN *status ) {
Catalog unnamed;
int Size;
OID  CAoid;

     // construct attribute list for internal type
     List attrList( &CatalogId, NEAR_FIRST_PHYSICAL, NULL, MAXATTR, status );
     check( status , " can not construct attribute list for unnamed" );

     Size = *offset ;
     strcpy( unnamed.TypeName, " ");
     setClassType( &unnamed.ClassType, 1 ); // user class ...
     HoodsAttr( Pattr -, offset, &attrList, status );
     check( status, "error creating attribute entries for unnamed" );

     attrList.getListOid( &CAoid, status );
     check( status., "can not get unnamed attribute list oid" );
     memcpy( &unnamed.List_Attributes, &CAoid, S_OID );
     memcpy( &unnamed.Statistics, &nullOid, S_OID );
     memcpy( &unnamed.List_SuperClass, &nullOid, S_OID );
     memcpy( &unnamed.List_SubClass, &nullOid, S_OID );
     memcpy( &unnamed.Instance_File, &nullFid, S_FID );
     unnamed.Size = *offset - Size ;
     writeType( unnamed, Ptypeoid, status );
     check( status, "can not write unnamed type" );

     .

};

void MoodsTypes::addAttributeToClass( char *PclsName, memberNode *Pmember, CatBOOLEAN *st
OID  typeOid ,atrOid;
Catalog typeinfo;
int      offset,l_size,l_type;

     getTypOid( PclsName, &typeOid, status );
     check( status, "can not find class entry" );

     readType( typeOid, &typeinfo, status );
     check( status, "can not read type info abuk");

     offset = typeinfo.Size;
     List attrlist( &typeinfo.List_Attributes, &l_type, &l_size, status );
          check(status, "can not construct list of attributes");

     MoodsAttr( Pmember, &offset, &attrlist, status );
     check( status, "can not add attributes to class");

     typeinfo.Size = offset;
     updateType( typeOid, typeinfo );
};


void MoodsTypes::MoodsAttr( memberNode *Attribute, int *offset, List *superList,
                            CatBOOLEAN *status ) {
AttributeType moodsattr;
int size;
OID unnamedOid,tmpOid;
Catalog typeinfo;

     if ( Attribute == NULL )  return;

     CatATTRIBUTE curAttr( AttributeId );
     strcpy( moodsattr.AttributeName , Attribute->memberName );
     moodsattr.AttributeOffset = *offset;
     memcpy( &moodsattr.Statistic, &nullOid, S_OID );
     switch ( Attribute->type ) {
             case TYPE_SET : {
                          moodsattr.AttributeConstructor = 'S' ;
                          size = *offset ;
                          createUnnamedType( Attribute->subAttributes, offset,
                                             &unnamedOid, status);
                          check( status, "error creating unnamed type ");

                          moodsattr.AttributeLength = S_OID ;
                          memcpy( &moodsattr.AttributeType, &unnamedOid, S_OID );
                          strcpy( moodsattr.AttrTypeName, " ");
                          moodsattr.ClassMember = 1 ;
                          curAttr.writeAttr( moodsattr, &tmpOid );
                          superList->insertToList( &tmpOid, status );
                          check( status, "error inserting super list" );
```

```
                      MoodsAttr( Attribute->attributeList, offset, superList,
                                 status );
                                                                      *offset =
                      check( status, "error creating unnamed type ");

                      break;
                      };
      case TYPE_LIST :{
                      moodsattr.AttributeConstructor = 'L' ;
                      size = *offset ;
                      createUnnamedType( Attribute->subAttributes, offset,
                                         &unnamedOid, status);
                      check( status, "error creating unnamed type ");
                      moodsattr.AttributeLength = S_OID  ;
                      memcpy( &moodsattr.AttributeType, &unnamedOid, S_OID );
                      strcpy( moodsattr.AttrTypeName, " ");
                      moodsattr.ClassMember = 1 ;
                      curAttr.writeAttr( moodsattr, &tmpOid ); *
                      superList->insertToList( &tmpOid, status );
                      check( status, "error inserting super list" );

                      MoodsAttr( Attribute->attributeList, offset, superList,
                                 status );

                      *offset = size + S_OID ;
                      check( status, "error creating unnamed type ");

                      break;
                      };
     case TYPE_REF : {
                      moodsattr.AttributeConstructor = 'R' ;
                      size = *offset ;
                      createUnnamedType( Attribute->subAttributes, offset,
                                       . &unnamedOid, status);
                      check( status, "error creating unnamed type ");
                      moodsattr.AttributeLength = S_OID  ;
                      memcpy( &moodsattr.AttributeType, &unnamedOid, S_OID  ;
                      strcpy( moodsattr.AttrTypeName, " ");
                      moodsattr.ClassMember = 1 ;
                      curAttr.writeAttr( moodsattr, &tmpOid );
                      superList->insertToList( &tmpOid, status );
                      check( status, "error inserting super list" );

                      MoodsAttr( Attribute->attributeList, offset, superList,
                                 status );
                      *offset = size + S_OID ;
                      check( status, "error creating unnamed type ");

                      break;
                      };
    case TYPE_TUPLE: {
                      moodsattr.AttributeConstructor = 'T' ;
                      size = *offset ;
                      createUnnamedType( Attribute->subAttributes, offset,
                                         &unnamedOid, status);
                      check( status, "error creating unnamed type ");
                      moodsattr.AttributeLength = *offset - size ;
                      memcpy( &moodsattr.AttributeType, &unnamedOid, S_OID );
                      strcpy( moodsattr.AttrTypeName, " ");
                      moodsattr.ClassMember = 1 ;
                      curAttr.writeAttr( moodsattr, &tmpOid );
                      superList->insertToList( &tmpOid, status );
                      check( status, "error inserting super list" );
```

```
                        MoodsAttr( Attribute->attributeList, offset, superList,
                                status );
                        check( status, "error creating unnamed type " );

                        break;
                        };
            default   :  {
                        moodsattr.AttributeConstructor = 'N' ;
                        size = *offset ;
                        getTypOid( Attribute->type, &tmpOid, status ) ;
                        check( status, "can not find type entry" );

                        memcpy( &moodsattr.AttributeType, &tmpOid, S_OID );
                        readType( tmpOid, &typeinfo, status );
                        check( status , "can not read type entry" );

                        strcpy( moodsattr.AttrTypeName, typeinfo.TypeName );
                        if ( Attribute->size) {
                            moodsattr.AttributeLength = Attribute->size ;
                            *offset = *offset + Attribute->size;
                        } else {
                            moodsattr.AttributeLength = typeinfo.Size ;
                            *offset = *offset + typeinfo.Size ;
                        };
                        moodsattr.ClassMember = 1 ;
                        curAttr.writeAttr( moodsattr, &tmpOid );
                        superList->insertToList( &tmpOid, status );
                        check( status , "can not insert to attribute list" );

                        MoodsAttr( Attribute->subAttributes, offset, superList,
                                status) ;
                        check( status, "error on creating internal type entries");

                        MoodsAttr( Attribute->attributeList, offset, superList,
                                status) ;
                        check( status, "error on creating internal type entries");
                        break;
                        };
        }; // case ends....
}; // MoodsAttr ENDS...


void MoodsTypes::createIndex( char *Pclass, OID Pindexoid , CatBOOLEAN *Pstatus) {
OID typeoid;
Catalog typeinfo;
int l_type, l_size;

    getTypOid( Pclass, &typeoid,Pstatus);
        check( Pstatus, "can not get type information while creating index");

        readType( typeoid, &typeinfo , Pstatus) ;
        check( Pstatus, "internal type read error" );

        List indxList ( &typeinfo.List_Index , &l_type, &l_size , Pstatus);
        check( Pstatus, "can not construct list of indexes");

        indxList.insertToList( &Pindexoid, Pstatus);
        check( Pstatus, "can not insert to index list");

        typeinfo.ifIndex = typeinfo.ifIndex + 1 ;
        updateType( typeoid, typeinfo );
};
```

```
void MoodsTypes::dropIndex( char *Pclass, OID Pindexoid, CatBOOLEAN *Pstatus ) {
OID typeoid,tmpoid;
Catalog typeinfo;
int l_type,l_size ,indx;


        getTypOid( Pclass, &typeoid,Pstatus);
    check( Pstatus, "can not get type information while creating index");

        readType( typeoid, &typeinfo , Pstatus) ;
        check( Pstatus, "internal type read error" );

        List indxList ( &typeinfo.List_Index , &l_type, &l_size , Pstatus);
        check( Pstatus, "can not construct list of indexes");

        for ( indx=0 ; indx < l_size ; indx++ ) {
                    indxList.getElement( &tmpoid, indx, Pstatus);
                    check( Pstatus, "internal: can not get element from index list"

                    if ( !memcmp( &Pindexoid, &tmpoid, S_OID ) ) {
                            // found.... so delete ...
                            cout << "index found and will be deleted n" ;
                            indxList.deleteFromList( indx,indx, Pstatus);
                            check( Pstatus, "can not drop index from index 1
                            typeinfo.ifIndex = typeinfo.ifIndex - 1;
                            updateType( typeoid, typeinfo );
                    return;
        };
    };
};



void MoodsTypes::addType( char * PclassName, LinkedList *Pinherits,
                    memberNode *Pmember, int User, CatBOOLEAN *Pstatus ) {
Catalog ClassHolder;
int err,attrOffset=0;
OID InheritsOid,SOid,CAoid,typeoid,indexoid;
void *my_entry;

  if ( !findType( PclassName ) ) { // no type name conflict..
      strcpy(ClassHolder.TypeName, PclassName);
      setClassType( &ClassHolder.ClassType, User ) ;
      List superList( &CatalogId, NEAR_FIRST_PHYSICAL, NULL, MAXATTR, Pstatus );
      check(Pstatus, "can not construct list");

              err = sm_CreateFile(BufferGroup, VolumeId, &ClassHolder.Instance_File );
              ErrorCheck(err, " error creating instance file ");

      List SuperClass( &CatalogId, NEAR_FIRST_PHYSICAL, NULL, MAXCLASS, Pstatus);
      check(Pstatus, "can not construct list");

              List indexlist( &CatalogId, NEAR_FIRST_PHYSICAL, NULL, MAXATTR , Pstatus
              check( Pstatus, "can not construct index list");

      indexlist.getListOid( &indexoid, Pstatus );
              check( Pstatus , "can not get index list oid");

              memcpy( &ClassHolder.List_Index , &indexoid , S_OID );

              ClassHolder.ifIndex = 0 ; // initially no index defined ..
```

```
if ( Pinherits != NULL ) { // there are some inherited classes ..
    // inherit from MoodsRoot first....

    getTypOid("MoodsRoot",&rootOid, Pstatus) ;

    if ( !(*Pstatus) ) { // no MoodsRoot is defined yet... so no inherit no.
        SuperClass.insertToList( &rootOid, Pstatus );
        check( Pstatus, "can not insert to list" );

        copyInheritedTypes( &rootOid, &superList, &attrOffset, Pstatus );
        check( Pstatus, "fail on inherits.." );
    } else  cout << "NO MOODSROOT CLASS DEFINED\n" ;

    my_entry = Pinherits->getFirstEntry();

    while ( my_entry != NULL ) { // do others...
            getTypOid(((struct inherits *)my_entry)->cname,&InheritsOid,Pstatus);
            check( Pstatus, "can not get info on type");

            SuperClass.insertToList(&InheritsOid, Pstatus);
            check( Pstatus, "can not insert to list");

            copyInheritedTypes( &InheritsOid, &superList, &attrOffset, Pstatus );
            check( Pstatus, "can not copy attributes of super classes");

            my_entry = Pinherits->getNextEntry();
    };

    SuperClass.getListOid( &SOoid, Pstatus );
    check( Pstatus, "error getting list oid" );

    memcpy( &ClassHolder.List_SuperClass, &SOoid, S_OID );

} else { // no inherited class ...

                    getTypOid("MoodsRoot", &rootOid, Pstatus);

                    if ( !(*Pstatus) ) { // ok , there is a MoodsRoot entry..
        SuperClass.insertToList( &rootOid, Pstatus );
        check( Pstatus, "can not insert to list");

        copyInheritedTypes( &rootOid, &superList, &attrOffset, Pstatus );
        check( Pstatus, "can not copy attributes of super classes");
    } else { cout << "NO MOODSTYPE CLASS\n" ; } ;

    SuperClass.getListOid( &SOoid, Pstatus);
    check( Pstatus , "error on getting list oid");

    memcpy( &ClassHolder.List_SuperClass, &SOoid, S_OID );
};

if ( Pmember != NULL ) {
    MoodsAttr( Pmember, &attrOffset, &superList, Pstatus );
    check( Pstatus , " error on creating attribute entries..." );
};
superList.getListOid( &CAoid, Pstatus );
check( Pstatus, "error getting attribute list oid");

memcpy(&ClassHolder.List_Attributes, &CAoid, S_OID );
ClassHolder.Size = attrOffset ;

List subclass( &CatalogId, NEAR_FIRST_PHYSICAL, NULL, MAXCLASS, Pstatus );
check( Pstatus , " error creating sub class list" );
```

```
        subclass.getListOid( &CAoid, Pstatus );
        check( Pstatus , "error getting list oid of subclass list");

        memcpy( &ClassHolder.List_SubClass, &CAoid, S_OID );

        createInstanceFile( &ClassHolder.Instance_File , Pstatus );
        check( Pstatus , "error creating instance file");


        memcpy( &ClassHolder.Statistics, &nullOid, S_OID );

        writeType( ClassHolder, &typeoid, Pstatus );
        check( Pstatus, "error writing type entry");

        SuperClass.getListOid( &SCoid, Pstatus );
        check( Pstatus , "error getting superclass list oid" );

        linkToSuper( SCoid, typeoid );
        *Pstatus = 0 ;
  } else *Pstatus = 1 ;
}; // END OF ADD TYPE ......

void MoodsTypes::printAllAtt( AttrList *inAtt )
{
        if( !inAtt ) return ;
        printf(" For the Attribute : \n");
        printf(" Name   : %s \n", inAtt->entry.AttributeName );
        printf(" Type   : %s \n", inAtt->entry.AttrTypeName );
        printf(" Offset : %d \n", inAtt->entry.AttributeOffset );
        printf(" Length : %d \n", inAtt->entry.AttributeLength );
        printf(" Cons.  : %c \n", inAtt->entry.AttributeConstructor );
        printf(" MEMber : %x \n", inAtt->entry.ClassMember );
        printf(" NEXT SUB \n ");
        printAllAtt( inAtt->nextsub );
        printf( " NEXT \n ");
        printAllAtt( inAtt->next );
        printf("-----------\n");
}
```

```
typedef struct {
        short returntype;
        char  fcnname[FUN_MAX_NAME];
        char  memberornot;
        char  clname[FUN_MAX_CNAME];
        char  RealName[FUN_MAX_SIGN];
        short argnum;
} CatEntry;

typedef struct {
        short rettype;
        char  fcnname[FUN_MAX_NAME];
        char  ismember;
        char  classname[FUN_MAX_CNAME];
        char  signature[FUN_MAX_SIGN];
        short argc;
} FunctionType;


int MoodsTypes::addFunc( FunctionType Pfunc, CatBOOLEAN *Pstatus) {
OID funcOid, typeOid, nullOid, listsOid;
int err;
Catalog typeinfo;

    if ( findType (Pfunc.classname) ) {
        // class exists ...
        getTypOid( Pfunc.classname, &typeOid, Pstatus );
        readType (typeOid, &typeinfo, Pstatus);
        if ( *Pstatus ) return(1) ; // error reading type;
        if ( memcmp( &typeinfo.List_subClasses, nullOid, sizeof(OID))  ) {
            *Pstatus = 1 ; return(1) ; // not accepted at this pt.
             // not allowed to add function to a class which is inherited
             // by another one..
        };
        // start search of function swignature...
        if ( search_signature ( Pfunc.signature ) ) {
            // function decleration does not exist...
            // insert function info & put Oid to function list of type..
            err = sm_CreateObject( BufferGroup,  &FunctionId,
                                   NEAR_FIRST_PHYSICAL, NULL, NULL,
                                   sizeof (FunctionType) , &Pfunc, &funcOid );
            ErrorCheck( err, "error occured creating function entry" );

            if ( memcmp( &typeinfo.List_functions, &nullOid, sizeof(OID) )) {
                // if no functions inserted to list before.. create new one...
                List funcList( &CatalogId, NEAR_FIRST_PHYSICAL, NULL,
                               MAXFUNC, Pstatus);
                if ( *Pstatus ) return(1); // error creating list...
                // we should update catalog entry....
                funcList.insertToList( &funcOid, Pstatus );
                if ( *Pstatus ) return(1); // error inserting to list...
                funcList.getListOid( &listsOid, Pstatus );
                if ( *Pstatus ) return(1); // error getting lists oid...
                memcpy ( &typeinfo.List_functions, &listsOid, sizeof(OID) ) ;
                // I should overwrite current catalog entry...
                err = sm_ReadObject(BufferGroup, &typeOid, 0, READ_ALL, &UserDesc );
                ErrorCheck( err, "reading type ");
                err = sm_WriteObject(BufferGroup, 0, sizeof(typeinfo), &typeinfo,
                                     &UserDesc, TRUE );
                ErrorCheck( err, "overwriting type" );
                // ok. function entry written & added to the catalog info of class..

            } else { // there is a function list in catalog entry...
                // reading function  list ...
                List funcList( &typeinfo.List_functions, &l_typ, &l_size, Pstatus );
                if ( *Pstatus ) return(1) ; // error reading function list..
                // update list ( insert new func info oid ) ..
                funcList.insertToList( &funcOid, Pstatus );
                if ( *Pstatus ) return(1) ; // error inserting list..
            }
        } else {
            *Pstatus = 1; // class does not exist...
        };
}; // END of addFunc.........

int MoodsTypes::search_signature( char *Psign ) {
OID funcid, nextid ;
FunctionType funcintern;
int err;
BOOL eof;

        err = sm_GetFirstOid ( BufferGroup, &FunctionId, &funcid, &ObjectHeader, &eof);
        ErrorCheck( err, "cna't getting first oid n" );

        if ( !eof ) {
            do {
                err = sm_ReadObject( BufferGroup, &funcid, 0, -1, &UserDesc );
                ErrorCheck (err, "error reading function entry" );

                memcpy( &funcintern, UserDesc->basePtr, sizeof FunctionType) );
                // if found return 1 ;;;
                if ( !strcmp(funcintern.signature, Psign) )  return(1) ;

                err = sm_ReleaseObject( &UserDesc ) ;
                ErrorCheck ( err, " on release object " );

                err = sm_GetNextOid ( BufferGroup, &funcid, &nextid,
                                      &ObjectHeader, &eof );
                ErrorCheck( err, "on getting next function's oid" );

                memcpy( &funcid , &nextid, sizeof (OID) ) ;
            } while ( !eof );
            return (0) ;
        };
}; // END of search_signature ...


int MoodsTypes::delType( OID Ptype, short forced, CatBOOLEAN *Pstatus ) {
Catalog Intern;
OID     element;

        readType( Ptype, &Intern, Pstatus );
        if ( *Pstatus ) return( 1 ); // can not read type info.

        if ( memcmp( &Intern.List_subclasses, &nullOid, sizeof(OID) ) ) {
            // no sub classes then delete class
            err = sm_DestroyObject( BufferGroup, &Ptype );
            ErrorCheck( err ,"error destroying object" );

        } else { // has sub classes ....
            if ( forced ) { // sure for delete..
                List sub_class( &Intern.List_subclasses, &l_typ, &l_size,
                                Pstatus );
                if ( *Pstatus ) return( -1 ); // error constr. list

                for (indx=0 ; indx < l_size ; indx++) {
                    sub_class.getElement( &element, indx, Pstatus );
                    if ( *Pstatus ) return( 1 ); // error getting element.

                    delType( element, forced, Pstatus ); // recurse...
                );
            } else  *Pstatus = -1 ; // warning code for having sub classes...
        );
);
```

```
#include <initialize.h>
extern int User;

void init_db(CatBOOLEAN *tmpStatus) {

FID tmpFid, CatalogFile;
OID tmpOid;
OID *nullOid;


//************** fill nullOid with all 0

   nullOid = (OID *)malloc ( sizeof (OID) ) ;

   memset( (char *)nullOid , 0 , sizeof( OID ) );

//*********************************************************
//******* create a database  & delete current db. *********

//******* if exists delete dbase...

{
Database  current(DATABASE, &tmpFid  , tmpStatus);
          if ( *tmpStatus == 0 ) {
                MoodsTypes old(tmpFid);
                old.deleteFiles();
                current.deleteDB( tmpStatus );
          };
          if ( *tmpStatus )
               cout << "database could not be deleted\n" ;
          else cout << "current database is destroyed\n" ;
          *tmpStatus = 0 ;
};

{ int dsize;
Database  newdb(DATABASE, &tmpFid , tmpStatus );
          if (*tmpStatus) cout << "error  creating database\n" ;
          else cout << "new database is created...\n" ;
          cout << "initialize\n" ;
          cout << "fid.page=" << tmpFid.pid.page << "volid=" << tmpFid.pid.volid ;
          cout << "\n" ;
          dsize = sizeof(FID);
          sm_GetRootEntry(VolumeId, DATABASE, &tmpFid, &dsize);
          cout << "reread of root entry\n" ;
          cout << "fid.page=" << tmpFid.pid.page << "volid=" << tmpFid.pid.volid ;
          cout << "\n" ;

//******* copy database (moodstypes (catalog) ) file id.

          memcpy( &CatalogFile, &tmpFid, sizeof( FID ) );
          *tmpStatus = 0 ;
};

Sequence newseq( SEQNAME, SEQ_INITAL, SEQ_MIN , SEQ_MAX, SEQ_CYC, SEQ_INC, tmpStatus);
cout << "sequence created\n" ;
//******* new database is created               ***********
//*********************************************************

//*********************************************************
//******  create catalog entries for basic types !! *******

{
MoodsTypes moods(CatalogFile) ;
          *tmpStatus = 0 ;

Catalog basic_types ;

//***** creation of INTEGER *****************************************
          strcpy( basic_types.TypeName, "INTEGER");
          basic_types.Size    = sizeof( int ) ;
          basic_types.ClassType = 'B' ;
          memcpy( &basic_types.List_Attributes, nullOid, sizeof( OID ) );
          memcpy( &basic_types.List_SuperClass, nullOid, sizeof( OID ) );
          moods.writeType( basic_types, &tmpOid, tmpStatus );
          memcpy( &Type_Store[basic_count++], &tmpOid, sizeof( OID ) );

          if ( *tmpStatus ) cout << "creation of INTEGER type failed\n" ;
          else cout << "INTEGER created.... \n" ;
          *tmpStatus = 0 ;
//***** creation of INTEGER completed *******************************

//***** creation of FLOAT *******************************************
          strcpy( basic_types.TypeName, "FLOAT");
          basic_types.Size    = sizeof( float ) ;
          basic_types.ClassType = 'B' ;
          memcpy( &basic_types.List_Attributes, nullOid, sizeof( OID ) );
          memcpy( &basic_types.List_SuperClass, nullOid, sizeof( OID ) );
          moods.writeType( basic_types, &tmpOid, tmpStatus );
          memcpy( &Type_Store[basic_count++], &tmpOid, sizeof( OID ) );

          if ( *tmpStatus ) cout << "creation of FLOAT type failed\n" ;
          else cout << "FLOAT created.... \n" ;
          *tmpStatus = 0 ;
//***** creation of FLOAT completed *********************************

//**** creation of BOOLEAN *******************************************
          strcpy( basic_types.TypeName, "BOOLEAN");
          basic_types.Size    = sizeof( char ) ;
          basic_types.ClassType = 'B' ;
          memcpy( &basic_types.List_Attributes, nullOid, sizeof( OID ) );
          memcpy( &basic_types.List_SuperClass, nullOid, sizeof( OID ) );
          moods.writeType( basic_types, &tmpOid, tmpStatus );
          memcpy( &Type_Store[basic_count++], &tmpOid, sizeof( OID ) );

          if ( *tmpStatus ) cout << "creation of BOOLEAN type failed\n" ;
          else cout << "BOOLEAN created....\n" ;
          *tmpStatus = 0 ;
//***** creation of BOOLEAN completed   *****************************

//***** create basic type CHAR **************************************
          strcpy( basic_types.TypeName, "CHAR");
          basic_types.Size    = sizeof( char ) ;
          basic_types.ClassType = 'B' ;
          memcpy( &basic_types.List_Attributes, nullOid, sizeof( OID ) );
          memcpy( &basic_types.List_SuperClass, nullOid, sizeof( OID ) );
          moods.writeType( basic_types, &tmpOid, tmpStatus );
          memcpy( &Type_Store[basic_count++], &tmpOid, sizeof( OID ) );
          if ( *tmpStatus ) cout << "creation of CHAR type failed\n" ;
          else cout << "CHAR created....\n" ;
          *tmpStatus = 0 ;

//***** creation of CHAR completed **********************************

//***** create basic type STRING ************************************
          strcpy( basic_types.TypeName, "STRING");
          basic_types.Size    = sizeof( char ) ;
          basic_types.ClassType = 'B' ;
          memcpy( &basic_types.List_Attributes, nullOid, sizeof( OID ) );
          memcpy( &basic_types.List_SuperClass, nullOid, sizeof( OID ) );
          moods.writeType( basic_types, &tmpOid, tmpStatus );
          memcpy( &Type_Store[basic_count++], &tmpOid, sizeof( OID ) );
          if ( *tmpStatus ) cout << "creation of STRING type failed.n" ;
          else cout << "STRING created.... n" ;
          *tmpStatus = 0 ;

//***** creation of STRING completed ********************************

//***** creation of TUPLE ********************************************
          strcpy( basic_types.TypeName, "TUPLE");
          basic_types.Size    = sizeof( OID ) ;
          basic_types.ClassType = 'T' ;
          memcpy( &basic_types.List_Attributes, nullOid, sizeof( OID ) );
          memcpy( &basic_types.List_SuperClass, nullOid, sizeof( OID ) );
          moods.writeType( basic_types, &tmpOid, tmpStatus );
          memcpy( &Type_Store[basic_count++], &tmpOid, sizeof( OID ) );

          if ( *tmpStatus ) cout << "creation of TUPLE type failed n" ;
          else cout << "TUPLE created.... n" ;
          *tmpStatus = 0 ;
//***** creation of TUPLE completed *********************************

//***** creation of SET **********************************************
          strcpy( basic_types.TypeName, "SET");
          basic_types.Size    = sizeof( OID ) ;
          basic_types.ClassType = 'S' ;
          memcpy( &basic_types.List_Attributes, nullOid, sizeof( OID ) );
          memcpy( &basic_types.List_SuperClass, nullOid, sizeof( OID ) );
          moods.writeType( basic_types, &tmpOid, tmpStatus );
          memcpy( &Type_Store[basic_count++], &tmpOid, sizeof( OID ) );

          if ( *tmpStatus ) cout << "creation of SET type failed.n" ;
          else cout << "SET created.... n" ;
//***** creation of SET completed ***********************************

//***** creation of LIST *********************************************
          strcpy( basic_types.TypeName, "LIST");
          basic_types.Size    = sizeof( OID ) ;
          basic_types.ClassType = 'L' ;
          memcpy( &basic_types.List_Attributes, nullOid, sizeof( OID ) );
          memcpy( &basic_types.List_SuperClass, nullOid, sizeof( OID ) );
          moods.writeType( basic_types, &tmpOid, tmpStatus );
          memcpy( &Type_Store[basic_count++], &tmpOid, sizeof( OID ) );

          if ( *tmpStatus ) cout << "creation of LIST type failed.n" ;
          else cout << "LIST created.... n" ;
//***** creation of LIST completed **********************************

//***** creation of REF **********************************************
          strcpy( basic_types.TypeName, "REF");
          basic_types.Size    = sizeof( OID ) ;
          basic_types.ClassType = 'R' ;
          memcpy( &basic_types.List_Attributes, nullOid, sizeof( OID ) );
          memcpy( &basic_types.List_SuperClass, nullOid, sizeof( OID ) );
          moods.writeType( basic_types, &tmpOid, tmpStatus );
          memcpy( &Type_Store[basic_count++], &tmpOid, sizeof( OID ) );

          if ( *tmpStatus ) cout << "creation of REF type failed.n" ;
          else cout << "REF created.... n" ;
//***** creation of REF  completed **********************************

//*********** ALL BASIC TYPES ARE CREATED AT THE MOMENT ************
```

```
***************
        cout << "creating MoodsRoot" ;
{
  FID atrfid;
  AttributeType Attrinfo ;
  OID atroid,listoid,tmptupleoid,atrlistoid;
  Catalog RootEntry,tmptuple;
          // define real attribute to be the empty type's attribute
          moods.getAttrFile( &atrfid );
          CatATTRIBUTE curAttr( atrfid );

          strcpy( Attrinfo.AttributeName, "TypeId" );
          moods.getTypOid( "INTEGER" , &Attrinfo.AttributeType, tmpStatus );
          strcpy( Attrinfo.AttrTypeName, "INTEGER" );
          Attrinfo.AttributeOffset = 0 ;
          Attrinfo.AttributeLength = sizeof (int);
          Attrinfo.AttributeConstructor = 'N';
          Attrinfo.ClassMember = 1 ;
          memcpy( &Attrinfo.Statistic , &nullOid, sizeof OID) );

          curAttr.writeAttr( Attrinfo, &atroid );

          // define dummy type's attribute list & insert attribute entry to it

          List atrlist( &CatalogFile, NEAR_FIRST_PHYSICAL, NULL, 10, tmpStatus);
          atrlist.insertToList( &atroid, tmpStatus );
          atrlist.getListOid( &listoid, tmpStatus);

          // define temprory class entry for root entry..

          strcpy( tmptuple.TypeName, " " );
          tmptuple.Size = sizeof(int);
          tmptuple.ClassType = 'C' ;
          memcpy(&tmptuple.List_Attributes , &listoid, sizeof(OID) );
          memcpy(&tmptuple.List_SuperClass , &nullOid, sizeof(OID) );
          memcpy(&tmptuple.Instance_File , &nullFid, sizeof(FID) );
          memcpy(&tmptuple.Statistics ,  &nullOid, sizeof(OID) );
          memcpy(&tmptuple.List_SubClass , &nullOid, sizeof(OID) );
          moods.writeType(tmptuple, &tmptupleoid, tmpStatus);

          // define actual root entry's attribute to be type...

          CatATTRIBUTE dummyattr( atrfid );
          strcpy( Attrinfo.AttributeName, " " );
          memcpy(  &Attrinfo.AttributeType, &tmptupleoid, sizeof(OID) );
          strcpy( Attrinfo.AttrTypeName, " " );
          Attrinfo.AttributeOffset = 0 ;
          Attrinfo.AttributeLength = sizeof (int);
          Attrinfo.AttributeConstructor = 'T';
          Attrinfo.ClassMember = 1 ;
          memcpy( &Attrinfo.Statistic , &nullOid, sizeof OID) );

          dummyattr.writeAttr( Attrinfo, &atroid );

          // create class information for moodsroot..

          List RootList( &CatalogFile, NEAR_FIRST_PHYSICAL, NULL, 10, tmpStatus);
          atrlist.insertToList( &atroid, tmpStatus );
          atrlist.getListOid( &atrlistoid, tmpStatus);

          List subcls(  &CatalogFile, NEAR_FIRST_PHYSICAL, NULL, 2000,  tmpStatus);
          subcls.getListOid( &listoid, tmpStatus);
          strcpy( RootEntry.TypeName, "MoodsRoot" );
          RootEntry.Size = sizeof(int);
        RootEntry.ClassType = 'C' ;
        memcpy(&RootEntry.List_Attributes , &atrlistoid, sizeof(OID) );
        memcpy(&RootEntry.List_SuperClass , &nullOid, sizeof(OID) );
        memcpy(&RootEntry.Instance_File , &nullFid, sizeof FID) );
        memcpy(&RootEntry.Statistics ,  &nullOid, sizeof(OID) );
        memcpy(&RootEntry.List_SubClass , &listoid, sizeof(OID) );
        moods.writeType(RootEntry, &tmptupleoid, tmpStatus );



      { FID atrfid;AttributeType Attrinfo ;
        OID atroid,listoid;
        Catalog RootEntry;
        moods.getAttrFile( &atrfid );
        CatATTRIBUTE curAttr( atrfid );
        strcpy( Attrinfo.AttributeName, "TypeId" );
        moods.getTypOid( "INTEGER" , &Attrinfo.AttributeType, tmpStatus );
        strcpy( Attrinfo.AttrTypeName, "INTEGER" );
        Attrinfo.AttributeOffset = 0 ;
        Attrinfo.AttributeLength = sizeof (int);
        Attrinfo.AttributeConstructor = 'N';
        Attrinfo.ClassMember = 1 ;
        memcpy( &Attrinfo.Statistic , &nullOid, sizeof(OID) );

        curAttr.writeAttr( Attrinfo, &atroid );

        List atrlist( &CatalogFile, NEAR_FIRST_PHYSICAL, NULL, 10, tmpStatus );
        atrlist.insertToList( &atroid, tmpStatus );
        atrlist.getListOid( &listoid, tmpStatus );

        strcpy( RootEntry.TypeName, "MoodsRoot" );
        RootEntry.Size = sizeof(int);
        RootEntry.ClassType = 'C' ;
        memcpy(&RootEntry.List_Attributes , &listoid, sizeof(OID) );
        memcpy(&RootEntry.List_SuperClass , &nullOid, sizeof(OID) );
        memcpy(&RootEntry.Instance_File , &nullFid, sizeof(FID) );
        memcpy(&RootEntry.Statistics ,  &nullOid, sizeof(OID) );

        List subcls(  &CatalogFile, NEAR_FIRST_PHYSICAL, NULL, 2000,  tmpStatus );
        subcls.getListOid( &listoid, tmpStatus);
        memcpy(&RootEntry.List_SubClass , &listoid, sizeof(OID) );
        moods.writeType(RootEntry, &atroid, tmpStatus);

      cout << " database is ready for operations "\n" ;

  };
  };
```

```
#include "Params.h"
#include "Types.h"
#include <malloc.h>
#include <memory.h>
#include <stdarg.h>

class Params {
       private:
         char *param_list;
         Integer  param_index[MAX_PARAMS];
         int  param_count;
       public:
         Params(int size) { param_list=malloc(size);param_count=0;param_index[0].value=0;
         void add_parameters(void * ptrval);
         void add_parameters(void * ptrval,int size);
         ~Params() { free(param_list); }
         char * get_parameters();
         Integer  * get_indexes();
};

void Params::add_parameters(void *ptrval) {
       memcpy(&param_list[param_index[param_count].value],ptrval,4);
       param_count++;
       param_index[param_count].value=param_index[param_count-1].value+4;

};

void Params::add_parameters(void *ptrval,int size) {

      memcpy(&param_list[param_index[param_count].value],ptrval,size);
      param_count++;
      param_index[param_count].value=param_index[param_count-1].value+size;

}

char *Params::get_parameters() {
     return(param_list);
};

Integer *Params::get_indexes() {
     return(param_index);
};
```

```
#include <stdio.h>
#include <string.h>
#include <malloc.h>
#include <memory.h>
#include "Dlfcn.h"
#include "Types.h"
#include "Functions.h"
#include "Params.c"

typedef struct  {
        int x;
        char *y;
 } my_struct;

class Function {
   private:
      char          class_name      [F_MAX_C_NAME_LEN];
      char          Function_name   [F_MAX_F_NAME_LEN];
      char          func_name_in_obj[F_MAX_FC_NAME_LEN];
      void          *parameters;
      int           indexes[F_MAX_PARAMS];
      int           types  [F_MAX_PARAMS];

      int           arg_count;
      void         *hand;

   public:

  Function(char *c_name,char *f_name, char *params, Integer *indx,
           Integer *typ,int arg,Integer  sz) ;
    int           error;
    void *        dynamic_link_func();
    int           search_Function();
    void          set_error();
    char        * convert_to_path();
    char        * find_f_name_in_obj();
                 ~Function() { dlclose(hand) ; };
};


int Function::search_Function() {
    return(1);
};

void Function::set_error() {
     printf("error occured in system n");
     error=0;
};

char *Function::convert_to_path() {
     return("..Shared");
};

char *Function::find_f_name_in_obj() {
     return("_resolve__F9my_structT17Integer5CharP");
};



Function::Function(char *c_name,char *f_name,char *params,Integer *indx,
 Integer *typ,int arg,Integer sz) {
 int i;
    error=1;
```

```
    arg_count=arg;                    /*  copy argument count to member */
    parameters=malloc(sz.value);
    memcpy(parameters,params,sz.value);
    for(i=0;i<arg_count;i++) {     /*  copy argument indexes and types to members */
       indexes[i]=((indx+i)->value);
       types[i]=((typ+i)->value);
    };

    strcpy(class_name,c_name);      /*  copy class name to member class_name */
    strcpy(Function_name,f_name);  /*  copy func  name to member func_name  */
    strcpy(func_name_in_obj,find_f_name_in_obj()); /* convert Function name into signatur
    if (!search_Function())         /*  search function signature in catalog */
         set_error();              /*  if no match , set error & return  */
};

void *Function::dynamic_link_func() {
int x=3,y=4;
void *retptr;

char class_object_name[F_MAX_PATH_NAME];

    strcpy(class_object_name,convert_to_path());


    hand=dlopen(class_object_name,1);
    if (hand==0) {
       set_error();
       exit(1);
    };

    void *(*func)(...);

    func=(void *(*)(...)) (dlsym(hand,func_name_in_obj));
    if (func==NULL) { printf("exiting .... n"); exit(3) ; };

    switch (arg_count) {
        case 1:
                  retptr=(*func) ( ( (char *)parameters+indexes[0] ) );
                  break;
        case 2:
                  retptr=(*func) ((( (char *)parameters+indexes[0] ) ),
                            (( (char *)parameters+indexes[1] ) ));
                  break;
        case 3:
                  retptr=(*func) ((  ( (char *)parameters+indexes[0] ) ),
                             (  ( (char *)parameters+indexes[1] ) ),
                             (  ( (char *)parameters+indexes[2] ) ));
                  break;
        case 4:
                  retptr=(*func) ((  ( (char *)parameters+indexes[0]   ),
                         (  ( (char *)parameters+indexes[1]   ),
                         (  ( (char *)parameters+indexes[2]   ),
                         (  ( (char *)parameters+indexes[3]   ));
                  break;
        case 5:
                  retptr=(*func) ((  ( (char *)parameters+indexes[0] ) ),
                             (  ( (char *)parameters+indexes[1] ) ),
                             (  ( (char *)parameters+indexes[2] ) ),
                             (  ( (char *)parameters+indexes[3] ) ),
                             (  ( (char *)parameters+indexes[4] ) ));
                  break;
     }
   return(retptr);
```

```
 };


int getsize(char *x) {
     int i=0;
     while(*(x+i)) i++;
     return(i);
};


main() {
    CharP       params;
    CharP       name;
    Integer     siz;
    Integer     oth;
    Integer     tp[3];
    my_struct mystruct;
    my_struct my2;
    my_struct *ret;
    Params parameter(24);


    mystruct.y=malloc(10);
    params.value= malloc(50);
    my2.y=malloc(10);
    name.value=malloc(20);

    strcpy(name.value,"iste boyle");
    oth.value=13;
    my2.x=40;
    mystruct.x=20;
    strcpy(mystruct.y,"tansel");
    strcpy(my2.y,"okay");

    siz.value=24;
    parameter.add_parameters(&mystruct,8);
    parameter.add_parameters(&my2,8);
    parameter.add_parameters(&oth,4);
    parameter.add_parameters(&name);

    Function x("resolve","resolve",parameter.get_parameters(),
                parameter.get_indexes(),tp,4,siz);
    if(x.error) {
         ret=(my_struct *)x.dynamic_link_func();
         printf("in main %d, %s\n",ret->x,ret->y);
    }
}
```