MoodView: An Advanced Graphical User Interface for OODBMSs

İsmailcem Budak ARPINAR Asuman DOĞAÇ Cem EVRENDİLEK
Software Research & Development Center
Scientific and Technical Research Council of Turkiye
Middle East Technical University
06531, Ankara TURKIYE

Abstract

OODBMSs need more than declarative query languages and programming languages as their interfaces since they are designed and implemented for complex applications requiring more advanced and easy to use visual interfaces. We have developed a complete programming environment for this purpose, called MoodView. MoodView translates all the user actions performed through its graphical interface to SQL statements and therefore it can be ported onto any object-oriented database systems using SQL with slight modifications.

MoodView provides the database programmer with tools and functionalities for every phase of object oriented database application development. Current version of MoodView allows a database user to design, browse, and modify database schema interactively and to display class inheritance hierarchy as a directed acyclic graph. MoodView can automatically generate graphical displays for complex and multimedia database objects which can be updated through the object browser. Furthermore, a database administration tool, a full screen text-editor, a SQL based query manager, and a graphical indexing tool for the spatial data, i.e., R Trees are also implemented.

Keywords: graphical user interfaces, object-oriented databases.

1. INTRODUCTION

MoodView* is the graphical front end to MOOD [8] which is a database system and an environment based on the object paradigm. MOOD implementation is realized on ESM (Exodus Storage Manager) [2] and has an object-oriented data model. An SQL like object-oriented query language called MOODSQL [8] is the uniform interface in accessing the database.

The main goals of such an interface and major issues for design and implementation of MoodView are:

Meeting OODBMS requirements: MoodView provides an environment which conforms to the requirements of object-oriented systems. It is a sophisticated, but easy to use interface. It provides visual access to the database without requiring expertise from the database user.

Full use of graphics: MoodView is based on the graphical direct manipulation paradigm and is implemented in C++ using X Windows/Motif toolkit which is a standard widget library on Unix machines.

Portability: MoodView produces SQL statements corresponding to the actions performed by the database user through the graphical user interface. Therefore, MoodView can be ported onto any object oriented systems using SQL with minor effort.

Compatibility: The community of database users need standard ways of defining data. MoodView does not ignore conventional interfaces such as C++ and SQL and integrates them into the graphical user interface. Class hierarchy of C++ code can be displayed visually and C++ code can be generated from the visual definitions.

Integrating query language into graphical user interface: MoodView provides a SQL based query formulation tool, namely MOODSQL, that the user can prepare her queries and receive results in a graphical

---

* The source code of MoodView is available from asuman@vm.cc.metu.edu.tr via e-mail.

environment.

Database design: MoodView provides a database browser to design, browse, and modify database schema interactively and display class inheritance hierarchy as a directed acyclic graph (dag).

Display generation for complex objects: MoodView provides automatic display for the objects in the database. These objects can have complex types or contain multimedia data and can be edited.

All-object principle: All the information maintained by MoodView is stored in the database catalog through the system defined classes. MoodView actions are performed through execution of the methods by the MOOD Kernel [8] on these data.

Furthermore, a graphical indexing tool for the spatial data and a database administration utility are also implemented.

In this paper, we describe the current version of MoodView, discuss the issues in the design and implementation. Section 2 presents the related work on OODBMS interfaces and gives a short description of some of those interfaces. Section 3 contains a simulation of a user session with MoodView and section 4 describes the design and implementation of MoodView.

2. RELATED WORK

Recently many OODBMSs with graphical user interfaces have been developed : *OdeView* [3] of ODE, *DBDesigner* [11] of ONTOS, and *tools* of O2 have visual interactive schema designers and database browsers. Furthermore, they provide various extensions. DBDesigner can generate C++ header files from the schema. OdeView and tools provide facilities for browsing objects. Tools is a front-end graphical programming environment, implemented as an O2 application and also contains a debugger. *O2 look* [4], an interface generator tool is used to visualize, edit, and walk through the database schema and instances. *Objectstore* also has a browser and a debugger. *Gemstone* has a programming environment provided by Smalltalk. *Facekit* of Cactis is a toolkit for designing graphical interfaces to object-oriented database systems [14].

Furthermore many research prototypes are also available such as *ADAM*, *RIDL*, *SNAP* [5], *Gambit*, *ISIS* [10], *DBDT*, *LID*, *SKI* [13], and *DDEW*. Some systems for constructing graphical user interfaces are also available such as *ET++* [24], *InterViews*, *MERCY*, and *Ingrid*. *ObjectWorks* and *GraphTrace* [15] are the examples of tools for aiding object-oriented programming.

There are research works to develop graphical query languages as user interfaces to databases [3, 6, 7, 12, 18].

Many other interface systems are proposed [9, 16, 20, 21]: *SIG* [19], *KIVIEW*, *GOOD* [22], *Object Display Definition System*, *G+*, *GraphLog*, and *GIUKU* [17] are examples of these systems.

We benefited from previous studies on user interfaces particularly from the ones implemented for the object-oriented database systems. Our display model is mainly inspired from the existing interfaces' navigation methods. But MoodView contributes to the research and development efforts in this area in the following respects: All tools for the management of data is integrated into GUI. Database user can define data through C++, SQL or MoodView and the definition can be visually displayed and visually defined data can be transformed to C++ code or SQL statements. The user can query database through a traditional query language SQL or through the graphical query facilities. Complex operations on objects is possible such as creation, deletion, update, projection, selection and automatic display generation for complex and multimedia objects. The user has various alternatives for managing data within this sophisticated environment and one can easily port it onto another OODBMS.

3. MoodView ENVIRONMENT

In this section we present a typical MoodView session with the help of MoodView screens' snapshots.

3.1. Initial Window and Starting the Server

Upon entering the programming environment, an initial window that contains the icons for each of the MoodView tools is displayed as shown in Figure 1(a).

Initial window contains a browser icon for both the class hierarchy browser and the object browser, a query icon for the query formulation tool and also an R Tree icon for the graphical indexing tool.

3.2. Database Design and Schema Updates

A database schema in MOOD contains class types, their

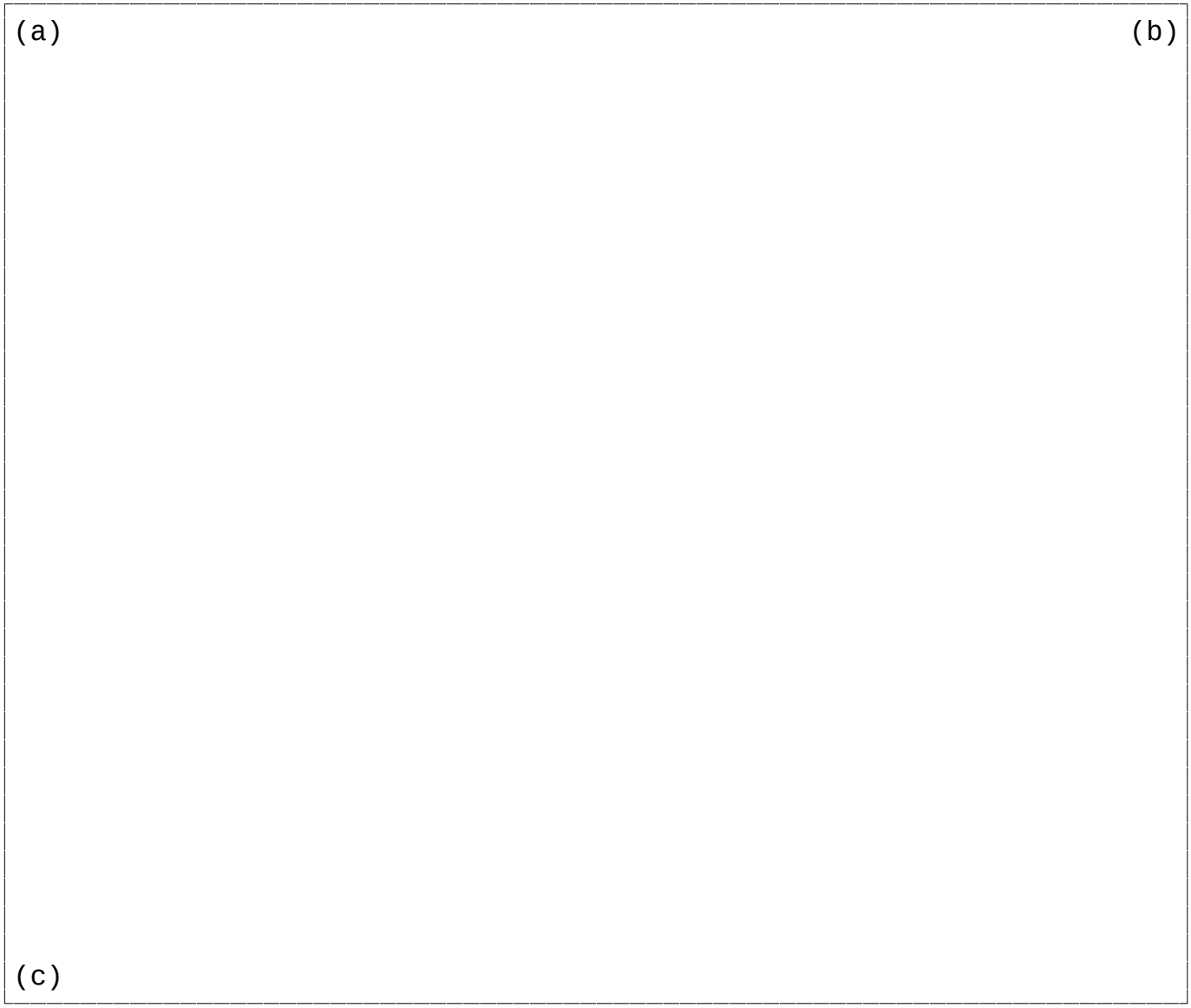

Figure 1. (a) Initial MoodView Window, **(b)** Data Defintion in C++, **(c)** Class Hierarchy Browser

methods and relationships between those classes. Their inheritance relationships is represented as a dag and MoodView uses a dag placement algorithm that minimizes crossovers and makes drawings for graph nodes [23].

3.3. Data Definition in C++

MoodView can display a class hierarchy defined in C++. We have modified cfront of C++ translator such that, when data is defined through C++, cfront extracts the schema information and stores into the MOOD catalog. MoodView uses the catalog information maintained by the MOOD kernel and displays class hierarchy graphically. Note that this provides for extensibility to our system since any DDL can be translated to corresponding C++ type definitions with minimal effort. MoodView also can convert graphically designed class hierarchy graph into C++ code.

Upon clicking on the browser icon in the initial window, MoodView displays the class hierarchy representing the database schema. All nodes representing the classes have standard menus activated by clicking on them. MoodView supports the primitive actions on the class hierarchy graph such as adding a new class, dropping an existing class, changing the name of a class etc.

Figure 1(c) shows the inheritance graph for a sample

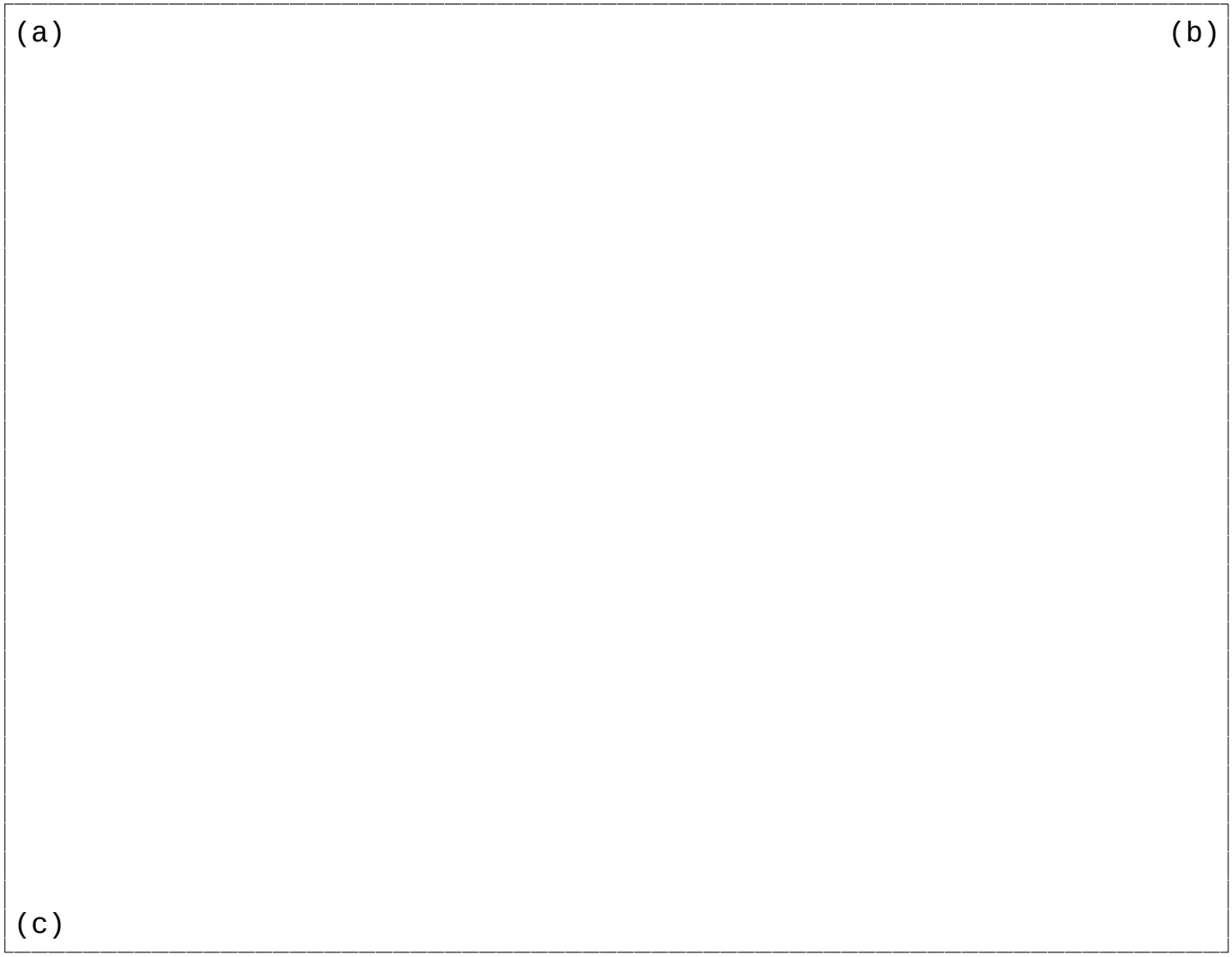

Figure 2. (a) Method Presentation, **(b)** Class Presentation, **(c)** Class Designer

database that belongs to a vehicle dealer.

3.4. Class Presentation

Assume that the car dealer starts selling cars and wants to extend his database to keep the information about every type of car he sells. He also decides to design "car class" as a specialization (subclass) of "vehicle class".

The car dealer designs the car class as follows:

```
class Car
{
    Type            string;
    Model           integer;
    Displacement    integer;
    Max_Output      integer;
    Max_Torque      float;
    Price           integer;
    Equipment_List  List Equipment;
    Photo           Ref Photograph;
}
```

Figure 1(b) shows the definition of the car class in C++ language. Note that the Equipment_List and Photo fields have complex types. Photo is a reference to a system defined class Photograph. The car dealer can add this new class to his database by selecting "Add subclass" item from the standard menu of Vehicle class presentation. This results in a pop-up template window which represents the new class. Each MoodView class presentation window contains a standard menu for schema updates, class type updates and class methods updates. MoodView class presentation shows a window that contains fields for the name of the class, its type id given by MOOD, its type constructor, superclasses, subclasses, and public and private methods and also a field that indicates if the class is a system or user

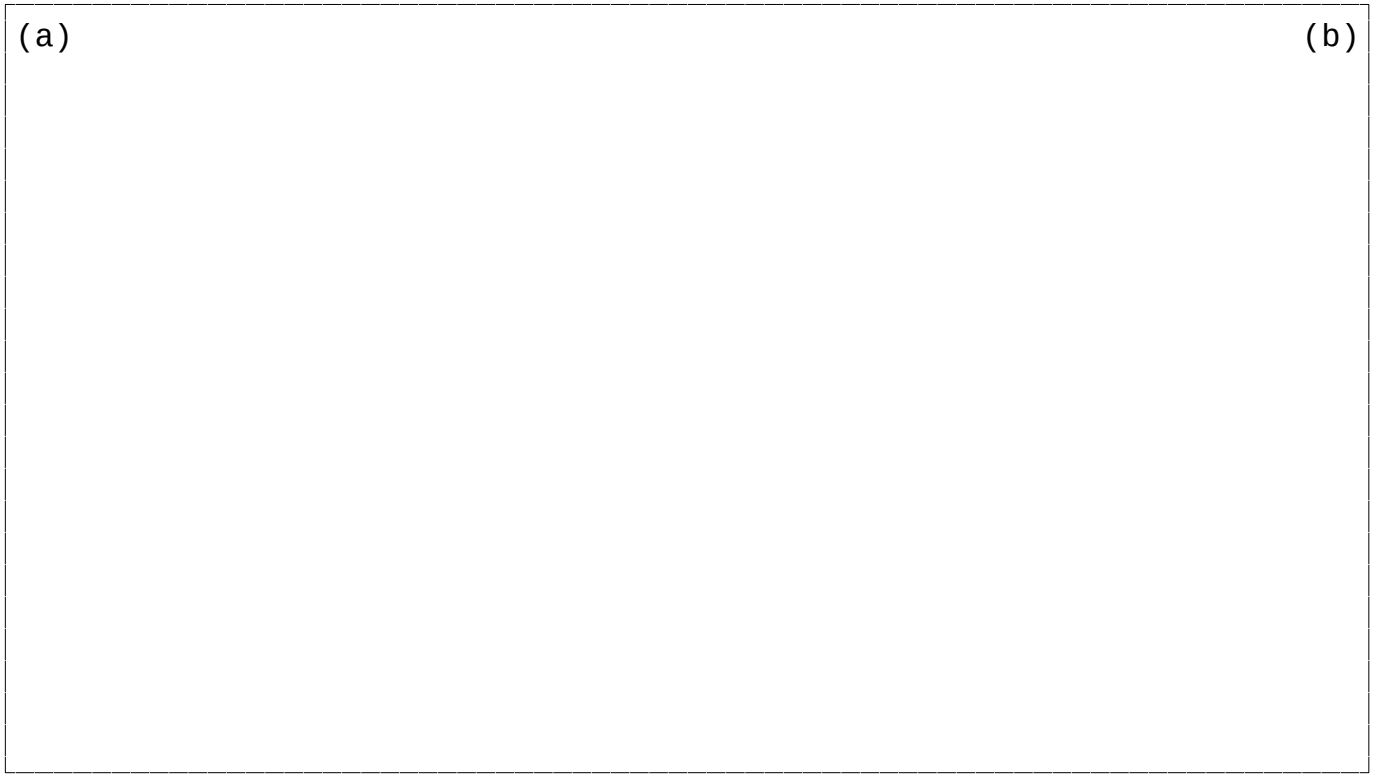

Figure 3. (a) Generic Presentation For A Car Object in SID Mode, **(b)** Car Objects Displayed in MID Mode

defined class as shown in Figure 2(b).

3.4.1. Class Design

Class attributes can be updated by clicking on attributes button. This is an entry point for a tool for designing object-oriented data types. One can add, drop attributes, change the name or the type of an attribute by using this tool (Figure 2(c)).

It is also possible to dump such class definitions to Unix files. At any time during the database design process, user can abort the transaction using the Cancel button on the presentation or commit the transaction and add the newly created class to database schema by clicking on Save button in the menu. After saving, the new class appears immediately on the class hierarchy window.

3.5. Method Presentation and Class Methods Updates

Now suppose that the car dealer wants to calculate each car's price in Turkish Liras. He can write a private method attached to the new class Car for the price calculation. Each class presentation of MoodView contains a menu for class methods' updates. Supported actions are adding a new method, dropping an existing method, changing the name of a method, changing the body and the parameters of a method. Class designer can call up the standard menu of the Car class and select the "Add Method" item.

A method template is used for the new method creation as shown in Figure 2(a). Updates to existing method bodies or creation of the new method bodies can be done through MoodView text editor.

3.6. Object Browsing

A user can access database objects through the query manager or through the object browser of MoodView. MoodView allows complex operations against a set of objects. These include creation, deletion, update and automatic display of complex and multimedia objects, and the invocation of methods. Projection, selection and complex query specification can be done on the objects through the SQL based query manager. Updates to objects from query displays are not allowed in MoodView.

3.6.1 Generic Object Presentations

Any complex type in MOOD can be created by using

basic types and recursive application of the type constructors (such as set, list or ref). Therefore MOOD objects constitute graphs connecting atoms and constructors and these graphs can be cyclic and large. MoodView has a generic display algorithm for displaying these object graphs and walking through the referenced objects. Referenced objects are represented as the drawn buttons and items of set and list are displayed in a scrolled window. Multimedia data such as images in different formats is defined through the system classes. As an example, the car presentation shown in Figure 3(a) contains references to a list of equipment objects and to the system defined Photograph class.

3.6.2. Display Modes: SID / MID (Single Instance Display / Multiple Instance Display)

We have two modes of displays for the objects of a class: Single Instance Display and Multiple Instance Display. MoodView can display a single object at a time (Figure 3(a)) or multiple objects in a scrolled window (Figure 3(b)). In the single instance window, the objects of a class can be sequenced by clicking on the arrow buttons. These modes are also used to display complex objects. A list is displayed in the MID mode whereas a set or ref is displayed in SID mode.

3.6.3. Updates to Object Presentations

Atomic types such as string or integer can be edited in the text mode. Complex types have different update semantics than the atomic types. A menu whose items correspond the update selections is placed on the widgets representing complex types. For example, the user can remove one of the equipments from the equipment list of the car or from the whole database by selecting the corresponding items from the update menu for the car object shown in Figure 3(a). Copy, paste operations are also allowed for both atomic and complex types. Query results can not be edited but one can copy an object from query display and paste it to the object browser. Dynamic type checking is performed by MoodView to ensure the correctness of updates.

3.6.4. Interactive Method Activation

Methods are attached to object presentations and can be activated interactively. For example, the car dealer can calculate the price of the car in Turkish Liras by clicking on pencil button representing applicable methods and selecting CalculatePrice from the menu.

3.7. Query Formulation

In MOOD, we have a uniform SQL-based interface in accessing the database. Query manager provides a query editor with facilities for accessing previous queries in a session. Through queries, objects with specific characteristics (selection) or selected portions of the objects (projection) can be displayed graphically. We are planning to implement an interactive query language that supports fast and easy formulation of object-oriented queries [6, 7, 12, 18]. This tool will provide the graphical construction of query predicates. Furthermore, a graphical query language to manipulate spatial data i.e. maps is also under development.

4. DESIGN and IMPLEMENTATION

MOOD Kernel interprets SQL statements and provides all the functions needed by MoodView to manage schema and instance levels as shown in Figure 4.

Kernel functions are divided between a SQL Interpreter and C++ compiler. The class methods compiled by C++ are used by the system through the Dynamic Function Linker [8].

4.1. MOOD Catalog

Kernel also performs catalog management. The catalog contains all of the information required to manage schema through MoodView such as 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 system defined classes MoodType, MoodAttribute and MoodFunction. MoodType class keeps track all the types used in the system. MoodAttribute stores the information about the attributes of these classes. The instances of the MoodFunction class keeps information about the member functions.

The design of MOOD Catalog makes MoodView easily extendible, so it can be used in a straightforward manner for new types and objects added to the MOOD. For example, MoodView uses this persistent type catalog to determine how an object of certain type is to be displayed. So, MoodView can generate automatic displays for any object in MOOD without requiring help from the user.

Figure 4. Overall MOOD Structure

4.2. SQL Interface Between Kernel and MoodView

One of the our major design goals is to provide portability for the graphical user interface. Therefore we have chosen a standard communication protocol between the database kernel and the graphical user interface. We avoided low level operations to interact with the database and all system specific operations are pushed to the kernel side. All the database operations performed by the user through MoodView are converted to SQL statements and the interpretation of SQL statements is performed by the kernel. SQL is a very common language in the database world. Therefore MoodView can be ported onto many database systems having an object-oriented data model and an SQL like query language. Some effort is needed because our SQL has some differences from the relational SQL because it contains object-oriented features such as the path expressions and the type constructors, i.e., set, list, ref.

MOOD has a client-server architecture. Both the kernel and the MoodView are on the client side of the system and the kernel is implemented as a collection of the primitives for database operations compiled into a shared library. Linking MoodView and the kernel together results in an executable client code. All the requests go to the server process (ESM) through a Remote Procedure Call like mechanism. The overall system structure is shown in Figure 4.

MOOD Kernel defines a class named as MoodViewManager that contains one method for the execution of SQL statements.

```
class MoodViewManager {
  // Local variables of MoodView
     .....
  // Method for Query Execution
     errorMessage executeQuery( ..... );
}
```

Whenever a user action requires a database operation at the schema or instance levels, MoodView passes the corresponding SQL statements to the kernel through executeQuery method. The function returns a message indicating the success or failure of the operation. The following examples are provided to clarify the communication protocol between the kernel and MoodView.

To create a new instance of the car class shown in Figure 3(a), MoodView produces the following SQL statement:

```
new Car < "Spring", 1993, 1400, 63, 10.5, 12000, {new Equipment
```

```
< "Radio-Tape", 250>},
        , SELECT p
          FROM Photograph p
          WHERE p.name = "IMG_DIR/Spring_Photo" >
```

To delete an item from the equipment list of the car, MoodView produces the following statement:

```
DELETE p[1]
FROM Car c, c.list p
WHERE c.name = "Spring";
```

4.3. Object Presentations

We have designed a cursor like mechanism which exists commonly in RDBMSs for displaying objects [1]. MoodView produces SQL statements corresponding to object requests from database through the object browser and the query manager. It is the kernel's responsibility to identify type and value of an object in the system at run-time using the MOOD Catalog. The kernel gets the stored representation of the object from the database and returns a pointer to a buffer area each element of which specifies a name, a type and a value of the object's attributes as shown in the following structures. MoodView synthesizes this information and combines widgets to display an object on the screen. Atomic types are displayed as the labeled text fields, and referenced objects are represented as the drawn buttons. Items of lists are displayed in a scrolled window.

```
typedef enum (REAL, INTEGER, TEXT, SET, LIST, REF,
              METHOD) TYPE;

class ATTRIBUTE {
public:
        char      name[MAX_ATTR_NAME_LEN];
        TYPE      attr_type;
        int       attr_width;
        int       attr_off;
        int       btree_avail;
        IID       index_id;  };

class BULK_SCHEMA {
        char          name[MAX_STRING_LEN];
        FID           fid;
        int           rec_width;
        BULK_TYPE     object_type;
        ATTRIBUTE     attr[MAX_ARITY];
        int           cardinality; };
```

It is possible to sequence through the returned objects using cursor primitives provided by the kernel. These primitives are listed below.

```
int OpenCursor (BULK_SCHEMA *InFile, Cursor *OutCur);
int GetNextObject(Cursor *Cur, USERDESC *outdesc, char *Flag);
int GetPrevObject(Cursor *Cur, USERDESC *outdesc, char *Flag);
int GetLastObject(Cursor *Cur, USERDESC *outdesc, char *Flag);
```

REFERENCES

[1] "ORACLE RDBMS Database Administrator's Guide, Version 6.0", Oracle Corporation, 1989.

[2] "Using the Exodus Storage Manager V2.1.1", June 1992.

[3] R. Agrawal, N. H. Gehani, J. Srinivasan, "OdeView: The Graphical Interface to Ode", Proc. ACM-SIGMOD Int'l Conf. on Management of Data, San Diego, California, 1990.

[4] P. Borras, J. C. Mamou, D. Plateau, B. Poyet, D. Tallot, "Building User Interfaces for Database Applications: The O2 experience", SIGMOD RECORD, Vol. 21, No. 1, March 1992.

[5] D. Bryce, R. Hull, "SNAP: A Graphics-based Schema Manager", IEEE Conf. on Data Eng., 1986.

[6] M. P. Consens, I. F. Cruz, A. O. Mendelzon, "Visualizing Queries and Querying Visualizations", SIGMOD RECORD, Vol.21, No. 1, March 1992.

[7] I. F. Cruz, "DOODLE: A Visual Language for Object-Oriented Databases", Proc. ACM-SIGMOD Int'l Conf. on Management of Data, 1992.

[8] A. Dogac, B. Arpinar, C. Evrendilek, T. Okay, C. Ozkan, "METU Object-Oriented DBMS" in Object-Oriented DBMSs, Springer Verlag NATO-ASI Series, in press.

[9] B. B. Flynn, D. Maier, "Supporting Display Generation for Complex Database Objects", SIGMOD RECORD, Vol.21, No. 1, March 1992.

[10] K. J. Goldman, S. A. Goldman, P. C. Kanellakis, S. B. Zdonik, "ISIS: Interface for a Semantic Information System", Proc. ACM-SIGMOD Int'l Conf. on Management of Data, Austin, Texas, 1985.

[11] S. Hong, J. Duhl, C. Harris, "DBDesigner: A Tool for Object-Oriented Database Applications", Journal of Database Administration, Summer 1992.

[12] H. J. Kim, H. F. Korth, A. Silberschatz, "PICASSO: A Graphical Query Language", Software Practice and Experience 18, 3(March 1988), 169-203.

[13] R. King, S. Melville, "Ski: A Semantics-Knowledgeable Interface", Proc. of the 10th Int. Conf. on VLDB, Singapore, August, 1984.

[14] R. King, M. Novak, "FaceKit: A Database Interface Design Toolkit", Proc. of the 5th Int. Conf. on VLDB, Amsterdam, 1989.

[15] M. F. Kleun, P. C. Ginarich, "GraphTrace - Understanding Object-Oriented Systems Using Concurrently Animated Views", OOPSLA' 88 Conf. Proc., San Diego, California.

[16] J. A. Konstan, L. A. Rowe, "Developing a GUIDE Using Object-Oriented Programming", OOPSLA' 91 Conf. Proc., Phoenix, Arizona.

[17] M. Kuntz, "The Gist of GIUKU Graphical Interactive Intelligent Utilities for Knowledgeable Users of Data Base Systems", SIGMOD RECORD, Vol.21, No. 1, March 1992.

[18] M. Kuntz, R. Melchert, "Pasta-3's Graphical Query Language: Direct Manipulation, Cooperative Queries, Full Expressive Power", Proc. of 15th Int. Conf. on VLDB, Amsterdam, The Netherlands, 1989.

[19] D. Maier, P. Nordquist , M. Grossman, "Displaying Database Objects", Proc. 1st Int'l Conf. Expert Database Systems, April 1986.

[20] J. C. Mamou, C. B. Medeiros, "Interactive Manipulation of Object-Oriented Views", Proc. of 7th Int. Conf. on Data Engineering, 1991.

[21] J. A. McDonald, A. Buja, "Painting Multiple Views of Complex Objects", ECOOP/OOPSLA' 90 Conf. Proc., Ottawa, Canada.

[22] J. Paredaens, J. V. Bussche, D. V. Gucht, V. Sarathy, L. Saxton, "An Overview of GOOD", SIGMOD RECORD, Vol. 21, No. 1, March 1992.

[23] K. Sugiyama, S. Tagawa, M. Toda, "Methods for Visual Understanding of Hierarchical System Structures", IEEE Transactions on Systems, Man ,and Cybernetics, Vol. Smc-11, No. 2, Feb. 1981.

[24] A. Weinand, E. Gamma, R. Marty, "ET++ - An Object-Oriented Application Framework in C++", OOPSLA '88 Conf. Proc., San Diego, California.