OdeView: The Graphical Interface to Ode

*R Agrawal**

AT&T Bell Labs
Murray Hill, New Jersey 07974

N H Gehani

AT&T Bell Labs
Murray Hill, New Jersey 07974

J Srinivasan

Purdue University
West Lafayette, IN 47907

ABSTRACT

OdeView is the graphical front end for Ode, an object-oriented database system and environment Ode's data model supports data encapsulation, type inheritance, and complex objects OdeView provides facilities for examining the database schema (i e , the object type or class hierarchy), examining class definitions, browsing objects, following chains of references starting from an object, synchronized browsing, displaying selected portions of objects (projection), and retrieving objects with specific characteristics (selection)

OdeView does not need to know about the internals of Ode objects Consequently, the internals of specific classes are not hardwired into OdeView and new classes can be added to the Ode database without requiring any changes to or recompilation of OdeView Just as OdeView does not know about the object internals, class functions (methods) for displaying objects are written without knowing about the specifics of the windowing software used by OdeView or the graphical user interface provided by it

In this paper, we present OdeView, and discuss its design and implementation

1 INTRODUCTION

OdeView is the graphical front end to Ode [1,2], a database system and environment based on the object paradigm Ode is an attempt to build a database system that offers one integrated data model for both database and general purpose manipulation The database is defined, queried, and manipulated using the database programming language O++, which is an upward-compatible extension of the object-oriented programming language C++ [28] O++ extends C++ by providing facilities suitable for database applications, such as facilities for creating persistent and versioned objects, defining and manipulating sets, organizing persistent objects into clusters, iterating over clusters of persistent objects, and associating constraints and triggers with objects

OdeView is intended for users who do not want to write programs in Ode's database programming language O++ to interact with Ode but instead want to use a friendlier interface to Ode OdeView is based on the graphical direct manipulation paradigm [26] that involves selection of items from pop-up menus and icons that can be clicked on and dragged OdeView provides facilities for examining the database schema (i e , the object type or class hierarchy), examining class definitions, browsing objects, following chains of references starting from an object, synchronized browsing, displaying selected portions of objects (projection), and retrieving objects with specific characteristics (selection)

OdeView does not need to know about the internals of Ode objects Consequently, the internals of specific classes are not hardwired into OdeView and new classes can be added to the Ode database without requiring any changes to or recompilation of OdeView OdeView uses dynamic linking to call class functions (member functions in C++ or methods), which exist in compiled form Each class must provide class functions for displaying objects of that class It is these special "display" functions that OdeView calls when objects are to be displayed OdeView interacts with member functions using a predefined protocol Just as OdeView does not know about the internals of the objects, the protocol for interacting with OdeView allows the class designer to write functions for displaying an object in one or

* Rakesh Agrawal is now with IBM Almaden Research Center San Jose CA 95120

more ways in different windows without knowing about the specifics of the windowing software used by OdeView or the graphical interface provided by it Thus objects can be displayed by different versions of OdeView which may be implemented quite differently, for example, these versions may be based on different windowing systems

In this paper, we describe OdeView, discuss the issues in the design and implementation of OdeView, and present extensions that we plan to implement in future Section 2 gives an overview of Ode Section 3 is a simulation of a user session with OdeView The design and implementation of OdeView is discussed in Section 4 Section 5 presents the design of some extensions that we are planning for OdeView Section 6 discusses related work and Section 7 summarizes our experience with OdeView

2. ODE OVERVIEW

The O++ object model is based on the C++ object model, called the *class* The class facility supports data encapsulation and multiple inheritance

An Ode database is viewed as a collection of persistent objects Persistent objects of the same type are grouped together into a *cluster*, the name of a cluster is the same as that of the corresponding type The database schema is the collection of class definitions of the objects that exist in the databases and the inheritance relationship between these types

3. A SAMPLE SESSION

OdeView is the graphical front end (user interface) to the Ode database system Currently, OdeView provides facilities to browse the database schema which in this case consists of the O++ class definitions and browse objects in the database

To give you a flavor of OdeView, we describe a sample session with OdeView

3.1 Initial Configuration and Schema Browsing

Upon entering OdeView, the user is presented with a scrollable "database" window containing the names and iconified images of the current Ode databases (Figure 1)

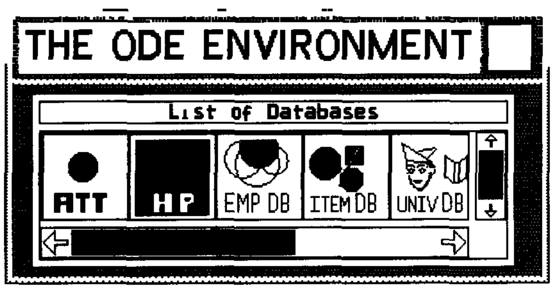


Figure 1 Initial Display

The user can select a database to interact with by using the mouse to click on the appropriate icon For example, let us look at the lab database identified by the ATT icon, this a small database about employees in our research center Upon clicking on the ATT icon, OdeView opens a 'class relationship" or schema window which displays the hierarchy relationship between the object classes in the ATT database (Figure 2)

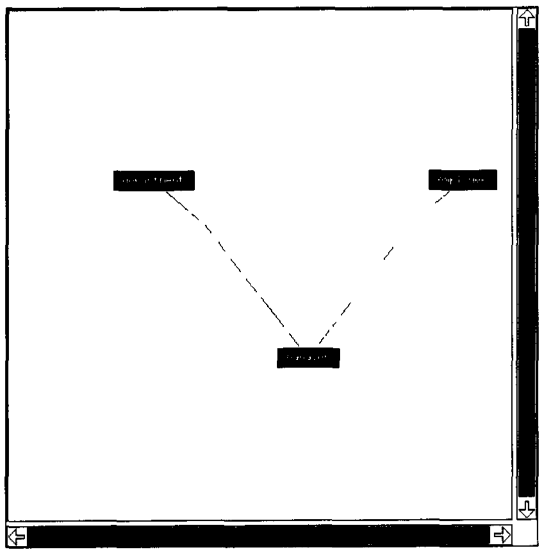


Figure 2 Lab Database

The hierarchy relationship between classes is a set of dags and OdeView uses a dag placement algorithm [19] that minimizes crossovers to display it

The user can zoom in and zoom out to examine this dag at various levels of detail The user can also examine a class in detail by clicking at the node labeled with the class of interest Clicking results in the opening of a "class information" window that has three scrollable subwindows, one for its superclasses the second for its subclasses and the third for the meta data associated with this class For example clicking on employee shows that it has no superclass one subclass manager, and that there are 55 objects in the employee cluster (Figure 3)

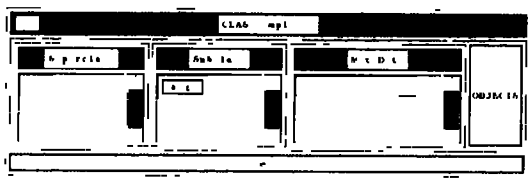
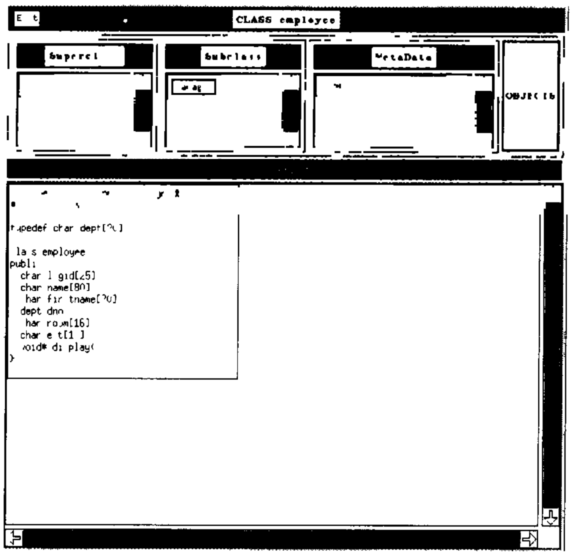


Figure 3 Class Information Window for Employee

The class information window also has a button, clicking which shows the class definition (Figure 4)



```
typedef char dept[?0]

 la s employee
publi
  char l gid[25]
  char name[80]
   har fir tname[?0]
  dept dno
   har room[16]
  char e t[1 ]
   void* di play(
}
```

Figure 4 Class Definition

The user may continue schema browsing by selecting another node in the schema graph, or may click on one of the superclasses or subclasses For example, clicking on manager opens up another window that shows that manager is the subclass of employee as well as department, that it has no subclasses, and there are 7 instances of managers (Figure 5) Browsing through the class information and relationship windows can be freely mixed

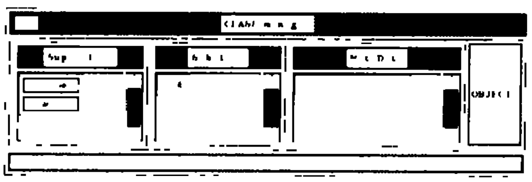


Figure 5 Class Information Window for Manager

3 2 Object Browsing

Associated with each class in Ode is a set of persistent objects of that class (this set is called a cluster) The class definition window has an "objects" button that allows users to browse through the objects in the cluster Clicking this button opens the "object set" window which consists of two parts the control and object panels The control panel consists of buttons reset, next, and previous to sequence through the objects The object panel has buttons to view the object

An Ode object can be displayed in one or more formats depending upon the semantics of the display function associated with the corresponding class This function is provided by the class designer The object set window supplies one button each for each of the object display formats For example, the employee object can be displayed textually or in pictorial form, the object panel for employee provides appropriate buttons to see these displays Figure 6 shows an employee object after the user has clicked the text and picture display buttons

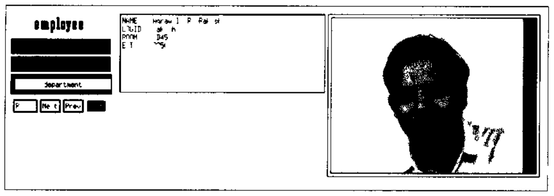


Figure 6 Employee Object

In this example, the display state of the employee objects consists of the object being displayed in both text and picture form OdeView remembers the display state of a cluster and will display other objects in the cluster in the same display state (until the user changes the display state, e g , by clicking the text button to close the text display)

3 3 Browsing Complex Objects

An object may contain embedded references to other objects The object panel of an object set window provides buttons for viewing these referenced objects For example, employee objects refer to department objects to view the department object associated with an employee, the user may click on the department button This opens up an ' object window[1] which contains buttons to view the referenced department object (Figure 7)

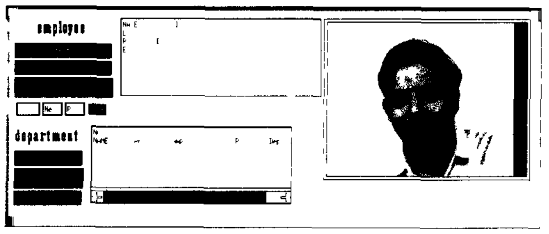


Figure 7 Employee's Department

Instead of containing a reference to a single object, an object may contain references to a set of objects For example, the department object may contain references to all the employees that work in the department To view employees who work in the same department, the user may click on the employees button This opens up an "object set" window consisting of an object panel to view an employee object and a control panel to sequence through the employees Figure 8 shows a colleague of rakesh working in the same department

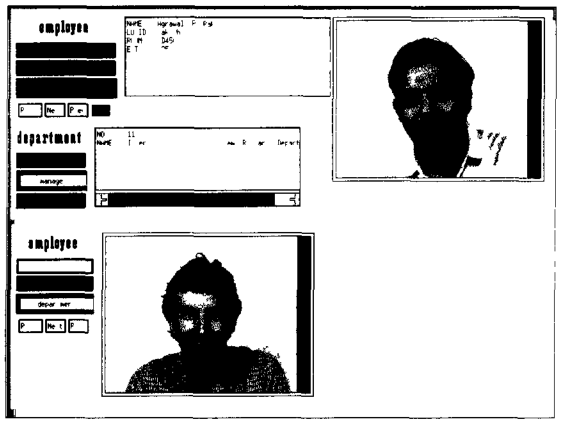


Figure 8 Employee's Colleague

3 4 Synchronized Browsing

The basic browsing paradigm encouraged by OdeView is to start from an object and then explore the related objects in the database by following the embedded chains of references To speed up such repetitive navigations, OdeView supports synchronized browsing [22] Once the user has displayed a network of objects and the user applies a sequencing operation to any object in this network the sequencing operation is automatically propagated over the network

For example suppose a user has set up the following display to view an employee's manager by following the chain of references that start from the employee (Figure 9)

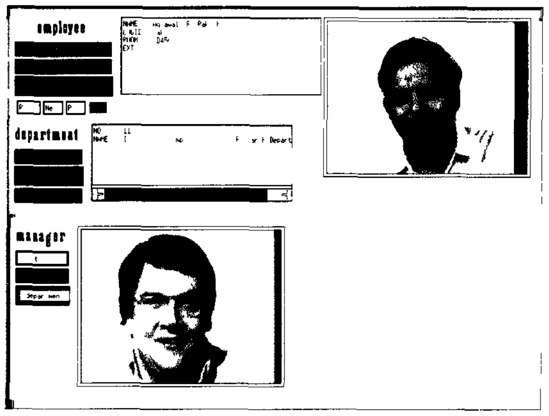


Figure 9 Employee's Manager

When the user looks at the next employee object the user would also want to see the next employee's department and manager In other words, we want the chain of displays starting with the employee object to be automatically updated in a "synchronized" fashion OdeView does exactly this For example clicking the next button of the employee object-set results in the following display (Figure 10)

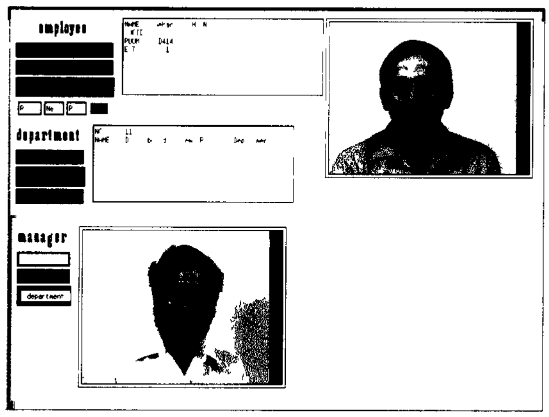


Figure 10 Synchronized Display

Note that we can be examining several databases and their schemas simultaneously, and within a schema we can be examining many classes and their objects simultaneously

4. DESIGN AND IMPLEMENTATION

We have built OdeView on top of the UNIX® operating system and using the X environment [25] We used HP-Xwidgets to build the graphical interface At present, we have completed the implementation of the browsing facilities, both at schema level as well as data level In this section, we discuss some of the interesting aspects of the OdeView implementation

The following are the major issues that we addressed in the OdeView implementation

- How an object should be displayed
- How to isolate OdeView from the specifics of classes and the class designer from the specifics of the windowing software
- How to insulate OdeView from the schema changes (addition, deletion, and modification of class definitions)

We now discuss these items in detail

1 An object window is the same as an object set window except that no sequencing buttons are provided because there is only one object to display

4.1 Displaying an Object

In a relational system, having selected an object (i e , a tuple), displaying it is rather straightforward it is simply a matter of displaying the attribute names which are available in the catalog and the corresponding values which are of simple types Attempting to automatically generate a display for Ode objects runs into the following problems

1 Ode objects are not simple tuples Their components can be of arbitrarily complex types they can be structures, arrays, sets, etc For some kinds of objects, it is possible to have fixed display schemes For example, nested structures may be displayed as indented values, sets as a list of values, and so on However, if these components represent more than just structured values, these schemes may not be adequate For example, it will be unsatisfactory to display a circular buffer simply as a linear list of values

2 Ode objects can contain references to other objects When displaying an object, how should the reference be displayed? As a simple value? Or should the referenced object be displayed?

3 Ode classes support data encapsulation The encapsulated data represents the implementation of the object, and a user sees only the public part of a class Therefore, if one respects encapsulation then only the public data and the data made available by calling the public member functions should be displayed But then how does one decide which member functions should be called? Simply calling all the public member functions will be unacceptable, if not potentially disastrous, because of any potential side effects

   Moreover, it may sometimes be beneficial to display the data in the private part, perhaps in a privileged mode, say for debugging This means that it should be possible to selectively violate the encapsulation

4 Depending upon the application, an object may be viewed differently or it may simply have multiple views For example, a document object may be viewed in text form, in Postcript form, or as a bitmap Thus, it should be possible to display an object in a variety of forms using one or more media

5 Some components of an object may have embedded semantics For example, suppose that one of the components of an object is a string that represents the name of the file containing some pictorial description of the object Displaying the string itself will not be of much value compared to displaying the pictorial representation which may require processing of

the pictorial description

In general, it will not possible for a system such as Ode to display an object without some help from the class designer The class designer knows best how an object is to be displayed Consequently, we decided that it is the responsibility of class designer to provide a distinguished display function (named `display`) for displaying objects of the class If the display function is not provided, then OdeView will synthesize a display function, possibly a rudimentary one Specifying a display function is similar to specifying constructor and destructor functions of a class in C++ for initializing objects when they are allocated and performing clean up when they are deallocated

4 2 Communication Protocol

Having decided that it is the responsibility of the class designer to supply a display function, we decided to adopt the following "principle of separation"

> The class writer should not have to know the specifics of object display (windowing) software and the display software should not have to know about object types

To realize this separation, we have defined a interface which is understood by both OdeView and the `display` function Specifically, we have defined a set of generic window types corresponding to the kind of windows that are supported by most windowing systems Some examples of window types are static text window, static text window with horizontal and vertical scroll bars, and raster image window These window types may be parameterized to allow the display function to choose the window sizes and to specify the relative placement between the windows The display function can choose any number of these window types to display an object in multiple ways

To display an object, OdeView calls the Ode object manager to the get the stored representation of the object into an object buffer Then OdeView calls the display function associated with the object class giving as an argument a pointer to the object buffer

As mentioned, an object can have multiple display representations The display function decides the window type for each display representation It also creates the display image for each display representation and puts it into in a display buffer The display function then returns an array of structures each element of which specifies a window type and a pointer to the corresponding display buffer

Using the array returned by the display function, OdeView creates appropriate windows and displays the object representations Subsequent scrolling, iconification, movement of these windows is handled by the underlying windowing software

To give you an idea of what the class designer is expected to write, we show the display function associated with the `employee` class [2]

```
#include "employee h"
#include <stdio h>
#include <string h>
void error(char *),
display_resources *employee  display()
{
    display_resources *drp, *p
    p = drp = new display_resources[NDISPLAYS],
    if ('drp)
        error("employee  display  out of store"),
    /*text window */
        p->kind = SCROLL_STATIC_TEXT,
        p->buf = new char [MAXSTRINGLEN],
        sprintf(p->buf,
          "NAME %s (%s)\nLOGID %s\nROOM %s\nEXT %s\n",
          name, firstname, logid, room, ext),
        strcpy(p->label, "text"),
    /*bit map window */
        ++p->kind = STATIC_RASTER,
        p->buf = new char [MAXSTRINGLEN], /*bitmap*/
        sprintf(p->buf, "%s/%s", PICDIR, logid),
        strcpy(p->label, "picture")
    /*oid for the department */
        ++p->kind = OID,
        p->buf = new char [MAXSTRINGLEN],
        sprintf(p->buf, "%ld", (long) deptoid),
        strcpy(p->label, "department"),
        strcpy(p->classname, "department"),
    /*thats it*/
        ++p->kind = 0
    return drp,
}
```

Note that the `employee` object contains a reference to the `department` object, but the display function associated with the `employee` class does not contain the code for displaying the `department` object, the display function associated with the `department` object is used for this purpose

Here is a fragment of the OdeView code for displaying an object of type `clname`

```
/*call object mgr to get display function loc */
    fn_location = get_dispfn(dbname, clname),
/*load the display function*/
    fn = ld_dispfn(fn_location),
/*get the first object in memory*/
    pobj = next(dbname, clname, FIRST),
/*call display function with pointer to object*/
    ptr_dres =  (display_resources *)(*fn)(pobj),
/*create windows and display object*/
    display_obj(ptr_dres),

/*wait for interrupt for next action  X loop*/
    XtMainLoop(),
```

4.3 Complex Objects

As mentioned in the previous section, if an object contains embedded references to other objects, then the panel for displaying objects of this type provides buttons for viewing the referenced objects These buttons are created in windows of type OID (object id), and the corresponding object id and the display function are associated with each such window When one of the above buttons is clicked, OdeView first calls the Ode object manager to get the corresponding object into an object buffer, and then calls the associated display function

4.4 Synchronized Browsing

When the user follows a chain of embedded references, a tree of windows is dynamically created This tree maintains the state of each window (open or closed) and pointers to display functions for windows of type OID When a sequencing action is performed at any node of this tree, the subtree rooted at this node is traversed recursively and data in the corresponding windows is refreshed Note that the refreshing is done irrespective of whether window is open or closed, as the user may open a window after performing the sequencing operation

4 5 Dynamic Linker

To display an object, OdeView calls the display function associated with the corresponding class Because OdeView is written in a compiled language, the simplest way of making the display functions of all the classes in the database available to OdeView would be to link all the display functions with OdeView However, this approach is unacceptable, given our desire to insulate OdeView from schema and class changes because this would require recompilation of OdeView every time the database schema is changed

The solution we used was to use dynamic linking which allows a compiled function to be linked to a load module Every time OdeView needs to display an object, it dynamically loads the object file containing the appropriate display function (if it is not already loaded) Dynamic linking is not part of the typical compiler based C and C++ environments, but we were able to scavenge the dynamic linker used in a C interpreter [17,23]

4.6 OdeView Process Structure

OdeView has been implemented as a collection of UNIX processes OdeView begins as a single process that allows a user to choose among different databases When the user selects a database, a "db-interactor" process is created that provides the interface for the user to interact with that database This scheme allows for simultaneous interaction with multiple databases Schema level operations such as viewing the class hierarchy as well as inspecting the class definitions are handled by the db-interactor process

When the user wishes to examine objects of a particular class, an "object-interactor" process is spawned This process dynamically loads and executes the display function defined by the class designer and also provides sequencing operations to scan all the persistent objects of that class The reason we chose to spawn a separate process to handle requests for interacting with objects of a specific class was to isolate effects of software failures The display function is the code written by the class designer If there are bugs in this code, then only the corresponding object-interactor process will be affected but not the whole OdeView Another advantage of using a separate process is that we can give the user the option of choosing where to place the window corresponding to that class (the X window system as an option allows a user to place a new UNIX process anywhere on the display) We experimented with the alternative approach of making OdeView decide where to place windows, but since it is impossible to predict the sequence of operations a user will perform, it turned out to be difficult to automatically generate good placements

Complex objects with embedded references to other objects are displayed in a "lazy" manner First only the top-level object is brought into the memory The display method is loaded and executed by passing a pointer to the top-level object If the object has constituent subobjects then the corresponding objects and the related display methods are loaded only if the user selects the appropriate buttons for seeing the subobjects

5. EXTENSIONS

At present OdeView supports only schema and object browsing We are now extending it to support projection and selection Our design for these features is discussed below

5.1 Projection

A straightforward scheme of allowing the user to project on any public member of a class will be inappropriate since the public members of a C++ class are not limited to being data elements or pure functions — they may be executable functions that do not return a value or cause side effects We, therefore, require that a class designer provide a function, `displaylist` that returns as its result the list of attributes on which projection can be performed A rudimentary `displaylist` display function is automatically synthesized if not explicitly provided by the class designer Note that these attributes may or may not correspond to the data members (public or private) of the class in question For example, an attribute to be displayed may actually be computed using other attributes

When a user wants to see a partial view of an object, the user clicks a "project" button that results in a set of buttons being created, one each for the displayable attributes of the object An ALL button is also created to allow projection on all attributes The user selects the projection attributes by clicking on the

2 The code has been modified slightly for the purposes of presentation

desired attributes

Internally, OdeView calls the `displaylist` function of the corresponding class, uses the list of attributes returned to create the buttons, and makes a bit vector corresponding to the attributes selected by the user The bit positions correspond to the positions of the attributes returned by `displaylist` As before, OdeView then calls the object manager to get a stored representation of the object in a buffer, and then invokes the corresponding `display` function

The `display` function will now be required to take an additional argument This argument is a bit vector representing the user's selection of attributes to be displayed The `display` function uses the bit vector argument to determine which attributes are to be shown in the object displays created by it If the bit vector argument is not supplied, then the `display` function uses a default bit vector (chosen by the class designer) to display the object The attributes displayed by default may be a subset or superset of attributes returned by the `displaylist` function Using the bit vector, the display function creates an appropriate display buffer and returns them to OdeView to display the object

5.2 Selection

When viewing objects in a cluster, a user may selectively view only a subset of the objects in the cluster In this case, the user may specify a selection predicate to select the objects to be displayed As in the case of projection, the user must be informed as to what attributes can be used to construct the selection predicate Our decision is to limit them to the attributes returned by the `selectlist` function A rudimentary `selectlist` display function is automatically synthesized if not explicitly provided by the class designer

Having displayed the list of attributes that can participate in a selection predicate, the user may use them to form the selection expression using a scheme similar to one proposed in [18] In this scheme, a predicate is formed by selecting from a menu of attribute names and operators and typing in values (or selecting values if there are only few of them) Another alternative is to use a condition box similar to QBE [35] and type in the selection condition as a string Our feeling is that the first scheme is desirable for simple selection predicates whereas the second scheme is preferable for complex predicates

Once OdeView has obtained the selection predicate, it passes the selection predicate to the object manager which uses it to filter objects retrieved from the databases Once an object has been selected and its memory copy is provided to OdeView, it is displayed using the implementation described earlier

5 3 Views involving more than one objects

A related issue is what should be displayed if the view involves a join of two or more objects This is a symptom of a general view definition problem in object-oriented database systems (see [12] for recent work) We have decided to display all the objects involved in the join simultaneously — each displayed using the corresponding `display` function

6. RELATED WORK

One of the first graphical display systems for databases was the Spatial Data Mangement System [13] An early browsing-by-navigation implementation was the entity-based interface presented in [6] Several other graphical interfaces for databases have been developed since then [3-5, 7-11, 14, 15, 18, 21, 24, 27, 29-34] A toolkit for designing graphics-based interfaces for object-oriented databases has been described in [16]

The design of OdeView was directly influenced by SIG [20], a system for generating displays of complex objects In SIG, associated with a class of objects is a *display type*, which is composed of one or more *recipes* Recipes correspond to different states of the object that are to be displayed A recipe has a *selection condition* that determines if the particular recipe applies, and *ingredients* that specify the positions, contents, and rendering of subregions of an display The display for an object is created by interpreting the associated display type In OdeView, we associate display functions with a class The effect of multiple recipes is accomplished by providing the projection capability to let the user create multiple views and letting the display function create multiple representations of an object OdeView supports display of richer objects in that the SIG objects do not have encapsulation and methods associated with them

An ingredient of a SIG's display type may also specify an *abstract view* Specifying an abstract view defers decisions on the display of a nested subobject to the display type associated with the subobject Otherwise, the display type of the object must include information for displaying the embedded subobject In OdeView, embedded objects are always displayed using the display functions associated with them

The navigation model of OdeView was inspired by the object-oriented browser, KIVIEW [22] KIVIEW was designed to interface with database models from which a semantic network may be extracted The design of KIVIEW supports browsing of objects in a class, exploration of related objects starting from an object, and the synchronized browsing

7 CONCLUSIONS

OdeView is designed to be a user-friendly graphical interface to Ode and is intended for users not wishing to write O++ programs OdeView can also be used to advantage by O++ programmers who need to understand the relationship between the different classes in the database

In building OdeView, we encountered three important issues which we feel we have addressed satisfactorily

1 How an object should be displayed OdeView will in general not know enough about the semantics of an object to display it properly, we therefore require each class designer to provide a display function which constructs object displays
2 How to isolate OdeView from the specifics of classes and the class designer from the specifics of the windowing software Isolating the two allows the class designer to write a display function that can be used by any graphical interface to Ode and it will not require the class designer to know the specifics of a particular implementation On the other hand, OdeView will not need to know about the specifics of each class to display objects of the class We solved problem by specifying a communication protocol to be used by OdeView and the display function to communicate with each other
3 How to insulate OdeView from the schema changes Since our environment is a compiled language, we did not want to recompile OdeView every time a schema changed We solved this problem by using dynamic linking

The initial reaction to OdeView has been favorable Users have found the following features particularly attractive

- Ability to mix schema browsing with object browsing and to do this simultaneously for more than one database
- Ability to explore the database by starting with one object and then examining related objects
- Use of synchronized browsing which speeds up exploration of the database by automatically refreshing displays of related objects when sequencing through a set of objects

8. ACKNOWLEDGMENTS

We appreciate the help of Doug Blewett who helped us understand X-Windows and HP Xwidgets and gave us code he had written to implement node widgets We are also thankful to Ted Kowalski and John Puttress for the dynamic linker, Steve North for the dag placement software, John Amanatides for the bitmap filter, and Don Mitchell for the bitmap scaling routines

REFERENCES

[1] R Agrawal and N H Gehani, 'ODE (Object Database and Environment) The Language and the Data Model'', *Proc ACM-SIGMOD 1989 Int l Conf Management of Data*, Portland, Oregon, May-June 1989, 36-45

[2] R Agrawal and N H Gehani, ''Rationale for the Design of Persistence and Query Processing Facilities in the Database Programming Language O++'', *2nd Int'l Workshop on Database Programming Languages*, Oregon Coast, June 1989

[3] D Byrce and R Hull, ''SNAP A Graphics-Based Schema Manager'', *Proc IEEE 2nd Int'l Conf Data Engineering*, Los Angeles, California, Feb 1986, 151-164

[4] D M Campbell, D W Embley and B Czejdo, ''Graphical Query Formulation for an Entity-Relationship Model'', *Data and Knowledge Eng 2*, (1986), 89-121

[5] T Catarci and G Santucci, ''Query By Diagram A Graphic Query System'', *Proc 7th Int'l Conf Entity-Relationship Approach*, Rome, Italy, 1988

[6] R G G Cattell, ''An Entity-Based Database Interface'', *Proc ACM-SIGMOD 1980 Int'l Conf on Management of Data*, 1980, 144-150

[7] I F Cruz, A O Mendelzon and P T Wood, ''A Graphical Query Language Supporting Recursion'', *Proc ACM SIGMOD Conf on Management of Data*, 1986, 16-52

[8] H Du and A Manoochehr, ''GQL A Graphical Database Language Using Pattern Images'', *Proc Computer Graphics Int'l Conf*, Geneva, May 1988

[9] R Elmasri and J A Larson, ''A User-Friendly Interface for Specifying Hierarchical Queries on an ER Graph Database'', *Proc 4th Int'l Conf Entity-Relationship Approach*, Chicago, Oct 1985, 236-245

[10] D H Fishman, D Beech, H P Cate, E C Chow, T Connors, J W Davis, N Derrett, C G Hoch, W Kent, P Lyngbaek, B Mahbod, M A Neimat, T A Ryan and M C Shan, ''Iris An Object-Oriented Database System'', *ACM Trans Office Information Systems 5*, 1 (Jan 1987), 48-69

[11] K J Goldman, S A Goldman, P C Kanellakis and S B Zdonik, ''ISIS Interface for a Semantic Information System'', *Proc ACM-SIGMOD 1985 Int l Conf on Management of Data*, Austin, Texas, May 1985, 328-342

[12] Heiler and S Zdonik Object Views Extending the Vision" *Proc IEEE 6th Int l Conf Data Engineering*, Los Angeles, California, Feb 1990

[13] C F Herot, "Spatial Management of Data', *ACM Trans Database Syst 5* 4 (Dec 1980), 493-513

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

[15] R King and S Melville, "Ski A Semantics-Knowledgeable Interface", *Proc of the 10th Int l Conf on Very Large Databases*, Singapore, Aug 1984, 30-33

[16] R King and M Novak, "FaceKit A Database Interface Design Toolkit", *Proc 15th Int'l Conf Very Large Data Bases*, Amsterdam, The Netherlands, Aug 1989, 115-123

[17] T J Kowalski, Y M Huang and H V Diamantidis, "Integrating Interpretive Technology into a Production Environment", *AT&T Technical Journal*, March/April 1990

[18] M Kuntz and R Melchert, "Pasta-3's Graphical Query Language Direct Manipulation, Cooperative Queries, Full Expressive Power", *Proc 15th Int l Conf Very Large Data Bases*, Amsterdam, The Netherlands, Aug 1989, 97-105

[19] R J Lipton, S C North and J S Sandberg, "A Method for Drawing Graph", *1st Symp Computational Geometry*, 1985

[20] D Maier, P Nordquist and M Grossman, "Displaying Database Objects", *Proc 1st Int'l Conf Expert Database Systems*, Charleston, South Carolina, April 1986, 15-30

[21] A Motro, "BAROQUE A Browser for Relational Databases", *ACM Trans Office Information Syst 4*, 2 (April 1986), 164-181

[22] A Motro, A D'Atri and L Tarantino, "The Design of KIVIEW An Object-Oriented Browser", *Proc 2nd Int'l Conf Expert Database Systems*, Tysons Corner, Virginia, April 1988, 17-31

[23] J J Puttress and H H Goguen, "Incremental Loading of Subroutines at Runtime", AT&T Bell Laboratories Technical Memorandum, Murray Hill, New Jersey, 1989

[24] L Rowe and K Shoens, "A Forms Application Development System", *Proc ACM-SIGMOD 1979 Int'l Conf on Management of Data*, Orlando, Florida, June 1982, 28-38

[25] R W Scheifler and J Gettys, "The X Window System' *ACM Trans Graphics 5* 2 (April 1986), 79-109

[26] B Shneiderman 'Direct Manipulation A Step Beyond Programming Languages', *IEEE Computer 16*, (1983) 57-69

[27] M Stonebraker and J Kalash, "Timber A Sophisticated Database Browser", *Proc of the 8th Int'l Conf on Very Large Databases*, Mexico City, Sept 1982

[28] B Stroustrup, *The C++ Programming Language*, Addison-Wesley, 1986

[29] F N Teskey, N Dixon and S C Holden, "Graphical Interfaces for Binary Relationship Data Bases", *Information Sys 3*, 2 (April 1984), 67-77

[30] Y Udagawa and S Oshuga, "Novel Technique to Interact with Relational Databases by Using Graphics Display", *J Information Processing 5*, 4 (1982), 256-264

[31] P Ursprung and C A Zehnder, "An Interactive Query Language to Define and Use Hierarchies", *Entity-Relationship Approach to Software Eng*, Elsevier Science Publishers B V (North Holland), 1983

[32] H K T Wong and I Kuo, "GUIDE Graphic User Interface for Database Exploration", *Proc of the 8th Int'l Conf on Very Large Databases*, Mexico City, Sept 1982, 22-32

[33] C T Wu, "A New Graphics User Interface for Accessing a Database", *Advanced Computer Graphics*, Springer-Verlag, 1986

[34] Z Q Zhang and A O Mendelzon, "A Graphical Query Language for Entity-Relationship Databases", *Entity-Relationship Approach to Software Eng*, Elsevier Science Publishers B V (North Holland), 1983

[35] M M Zloof, "Query-By-Example Operations on the Transitive Closure", RC 5526, IBM, Yorktown Hts, New York, 1975