ABSTRACT

Implementation of an Object-Oriented University Database Using VODAK/VML Prototype-2/C++

by

Bheeman Lingan and Madhumathi Tulasiram

Object–oriented database (OODB) technology has become very popular and successful in recent years. Currently, there are many commercial object–oriented database systems available that are used for developing large and complex real world applications. In addition, there are many research prototypes of object–oriented databases available. In a joint research project of the CIS Dept at NJIT and GMD-IPSI an object–oriented data model called the Dual Model was developed. Using this Dual Model a university environment database schema was designed. This university database schema was implemented using the VODAK/VML OODB prototype – 1. The university database was then reimplemented using the C++ programming language, without any underlying database.

In this thesis we have implemented a Dual Model based version of the university database, using the VODAK/VML OODB prototype – 2. This prototype is the first implemented prototype, which separates the structural and semantic aspects of a class definition. It also uses C++ as implementation language, while the previous prototype was based on Smalltalk – 80. To interact with the database we have developed a university database browser using the X11/MOTIF toolkit and C++.

IMPLEMENTATION OF AN OBJECT-ORIENTED UNIVERSITY DATABASE USING VODAK/VML PROTOTYPE-2/C++

by

Bheeman Lingan

and

Madhumathi Tulasiram

A Thesis
Submitted to the Faculty of
New Jersey Institute of Technology
in Partial Fulfillment of the Requirements for the Degree of
Master of Science in Computer Science

Department of Computer and Information Science

May 1993

APPROVAL PAGE

Implementation of an Object-oriented University Database Using VODAK/VML Prototype-2/C++

Bheeman Lingan and Madhumathi Tulasiram

5/10/93

Dr. Yehoshua Perl, Thesis Advisor (date)
Professor of Computer and Information Science,
New Jersey Institute of Technology

5/10/93

Dr. James Geller, Thesis Co-Advisor (date)
Assistant Professor of Computer and Information Science,
New Jersey Institute of Technology

Dr. Aaron Watters, Committee Member (date)
Assistant Professor of Computer and Information Science,
New Jersey Institute of Technology

BIOGRAPHICAL SKETCH

Author: Madhumathi Tulasiram

Degree: Master of Science in Computer Science

Date: Jan 1994 (expected)

Date of Birth:

Place of Birth:

Undergraduate and Graduate Education:

- Master of Science in Computer Science
  New Jersey Institute of Technology, Newark, NJ, 1994 (expected)
- Master of Science in Astronomy and Astrophysics,
  Osmania University, Hyderabad, India, 1987

Major: Computer Science

This work is dedicated to my wife Parvathi and my daughter Chandni.

Bheeman Lingan

This work is dedicated to my parents and my friend Moti Vyas.

Madhumathi Tulasiram

ACKNOWLEDGMENT

We would like to express our sincere gratitude to our advisor, Prof. Yehoshua Perl, CIS Department, NJIT, for his friendly advice and invaluable guidance and contributions throughout this thesis. We would also like to thank Dr. James Geller, our co-advisor, for all good suggestions he made regarding this thesis. We would like to thank our committee member Dr. Aaron Watters.

Ashish Mehta deserves much credit for his technical and moral support throughout this work. Despite his busy schedule he spent enormous amounts of time discussing various aspects of the project which has made this work possible.

We would like to thank all the members of our research group, those who have participated in the development of the University Database, and those who have participated in the development of OODINI.

We also extend our sincere thanks to the Chairman and Faculty of the Computer Science Department. Last but not the least, we want to thank all our friends who encouraged us and those who gave tips which helped us to complete our thesis work.

PREFACE

This thesis is the result of a joint project of the two authors. Bheeman worked on the development of the graphical schema and the development of the user interface using C++ and Motif. Madhumathi worked on the compilation of the schema, writing the application in C++, and all the required methods in VML. Chapters 2 and 4 represent the work of Madhumathi and Chapters 3 and 5 represent the work of Bheeman.

TABLE OF CONTENTS

Chapter Page

LIST OF FIGURES

CHAPTER 1

INTRODUCTION

Object-oriented databases (OODBs) have recently become very popular. There are several commercially available OODBs, such as, Gemstone [BOS91], Objectstore [OHMS92], ONTOS [M91], etc. In addition, many research efforts resulted in OODB prototypes, for example, VODAK/VML.

In a research project at NJIT and GMD-IPSI an OODB model, called the Dual Model [NPGT91, GPN91, NPGT90, NPGT89] has been developed. One of the distinguishing features of the model is that it separates structural and semantic aspects of a class definition. Using this Dual Model, a university environment OODB schema was created[CT90]. This schema contains around 180 classes. It can be considered a medium sized and complex schema and can be used to check the functionality of modeling with the Dual Model and to experiment with the new technology of Path Method Generation [M93].

The VODAK/VML prototype – 2 [KNBD92] is an OODB research prototype developed at GMD–IPSI, Darmstadt, Germany. VODAK is an object–oriented data model and VML (= VODAK Modeling Language) is a modeling language for it. This system has been developed using the C++ programming language [S91]. The current VODAK/VML prototype – 2 supports the Dual Model, i.e., it allows the separation of structural and semantic aspects of a class definition. To our knowledge this is the

only existing OODB prototype which allows such a distinction.

We have implemented the university database using VODAK/VML prototype – 2. As a basis for this work we have used the university environment database schema developed in [CT90]. The graphical representation of the schema was built using OODNI, the graphical schema representation system [HPGN92]. The Code Generator [C92] supported by OODINI, generates VML code from a displayed schema. Still, one needs to edit this code due to several reasons which we will discuss later.

Once the schema is edited such that it can be compiled by the VML compiler, methods can be added to each class. Such methods were not defined in [CT90], but they are necessary for the university environment database using VODAK/VML and were written for this research project. Then we wrote an application program using VML as well as C++ to access the university database.

We also developed a university database browser, using the X11/MOTIF toolkit and C++, as a graphical user interface for the database. The purpose of this university database browser is to allow the user to access information from the database interactively and easily. Using this university database browser the user can create a new database from scratch, and enter, update, delete and access all the information in the database.

We now briefly summarize the overall research project at NJIT and GMD-IPSI which is continued by this thesis. Previously, the VODAK/VML prototype – 1 was developed at GMD–IPSI, using Smalltalk–80 [GR83]. The VODAK/VML prototype

– 1 was not based on the Dual Model. The university database was developed using this prototype in a sequence of releases [K90, B90, D91, P91a]. First, the schema of [CT90] was changed to fit this prototype [K90, B90] and the preliminary version of the university database was developed. Then in [D91] an enhanced version of this university database was developed, which supported all the methods required for the database browser of this implementation. Then in [P91a] a database browser using the notion of MVC (=Model–View–Controller) supported by the Smalltalk–80 system, was developed. In [P91b] an implementation of the the university database using the C++ programming language was discussed.

This report is organized as follows: Chapter 2 discusses some background of the thesis, which includes the Dual Model, the university OODB, and the previous implementation of the university database. In Chapter 3 we discuss the graphical schema representation of this university OODB using OODINI. Chapter 4 discusses implementations of the university environment OODB using the VODAK/VML prototype. The basic features of this prototype, how to compile a schema, how methods are added, and how to compile and run an application using this prototype are outlined. In Chapter 5 we discuss the implementation of a university database browser using the X11/MOTIF toolkit and C++. There we also discuss the notion of MVC for X11/MOTIF, C++ classes used for implementing the database browser, screen layout for the database browser, and integration of the browser with the VODAK/VML OODB.

CHAPTER 2

BACKGROUND

2.1 The Dual Model for OODB

The Dual Model is an object-oriented data model, which draws a clear distinction between structural and semantic aspects of a class definition. Most object-oriented data models which are currently available do not support such a distinction. Thus, for such OODB models the structural and semantic aspects are intermingled, and they lack accurate representation of the application. On the other hand, the Dual Model which allows better abstraction due to this distinction, enables more accurate representation of the application. The structural aspects of a class are specified by an abstract data type called the object type of the class. The semantic aspects are specified by the object class. Every class description in the Dual Model can have four properties.

1. *Attributes* specify printable values of a given data type.

2. *User defined relationships* are pointers to other classes

3. *Methods* specify operations that can be applied to instances of a given class.

4. *Generic relationships* are system defined connections between classes. There are three structural generic relationships viz. *setof, memberof, subtypeof*, and

two semantic generic relationships supported by the Dual Model, *categoryof* for two classes which are in the same context and *roleof* for two classes which are in different contexts.

For a more detailed discussion of the Dual Model refer to [NPGT91, NPGT90, NPGT89, GPN91]. The following is a class definition of person.

class person
objecttype: PERSON
attributes:
PerData: PERDATATYPE
SocialSecNr: INT
Address: ADDRESSTYPE
Telephones: TELEPHONESTYPE
VisaStatus: STRING
relationships:
Father: person
Mother: person

The class **person** has five attributes *PerData, SocialSecNr, Address, Telephones* and *VisaStatus.* Another class is the class **instructor**, which has four attributes and is shown below. The class **instructor** is *roleof* class **person**.

class instructor
objecttype: INSTRUCTOR subtypeof: PERSON
roleof: person
attributes:
Teacheval: REAL
SocialSecNr: INTEGER
OfficeAddress: COMPANYADDRESSTYPE

Department: DEPARTMENTTYPE
essential: SocialSecNr
relationships:
Teaches: sections

The object type INSTRUCTOR is a subtype of the object type PERSON, and has the relationship *Teaches* to the class **sections**. The essential property *SocialSecNr* specifies that each instance of the class **instructor** has a non-nil value for *SocialSecNr*. The definition of the class **section** is not given here but is shown in the appendix. Generally, all the object types are defined first. Then classes are defined. The advantage is that many classes can share the same object type. For example, the object type STUDENT, shown below, is shared by two classes, **student** and **formerstudent**.

objecttype STUDENT
subtypeof: PERSON
memberof: STUDENTS
attributes:
StudentId: INT;
Degree: STRING;
LocalAddr: ADDRESSTYPE;
relationships:
Membership: ORGANIZATION
Transcript: TRANSCRIPT;

class student
objecttype: STUDENT
roleof: person
memberof: students
essential: Studentid
relationships:
Membership: union

class formerstudent

objecttype: STUDENT
roleof: person
memberof: formerstudents
relationships:
Membership: aluminorganization

Note that the class **student** has the relationship *Membership* to the class **union**, while class the **formerstudent** has the relationship *Membership* to the class **alumniorganization**. Both the classes **union** and **alumniorganization** share the object type ORGANIZATION. This object type definition is not shown here.

2.2 University Environment OODB

In our research project a university environment OODB has been modeled [CT90]. We will describe classes of that schema briefly. To summarize our discussion, we have grouped classes of this university environment. The VML code for all the classes is shown in the Appendix A. Currently the university schema contains about 180 classes.

Let us start our discussion with the **student** related classes where the class **student** is *roleof* person. There are several specialization classes of the class **student**, such as **undergrad, grad, formerstudent, matriculate,** etc.

In the **course** related classes, we define the class **transcript**. It has a relationship to the class **course_records**, which contains information about courses already completed by a student. Other classes of this group are **crsections, sections,** etc.

Next, we shall describe classes related to **instructors** who teach those courses, such as **professor, facultymember, visitprofessor, senior professor,** etc.

A graduate student can also be an **assistant**. An assistant can either be a teaching assistant or a faculty assistant or a research assistant.

Now, we will describe the university-related classes. Each university has many colleges. Each college has many departments. University related classes are **school, college, department,** etc. Now, let us consider another group of classes which are related to academic_appointments. We will describe classes related to **academic appointment**. The president is in charge of the university. Other related classes of academic appointment are **college_dean, vice_president, dept_chairperson,** etc.

Each **student** and **employee** has a **resume**. We define resume related classes such as **formal_education, informal_education, teaching_activities, publications** etc. There is another group of classes related to the staff of the university, like **student_staff, permanent_staff, department_staff, admin_staff,** etc.

A university environment has many **committees**. We have defined several classes to specify such committees, e.g., **acad_committee, univ_committee**, and **grad_committee**.

Finally, we have a few classes describing **computers** and **academic facilities**.

2.3 Previous Implementation of University OODB

The university database schema has been developed using the Dual Model. The

first release of VODAK/VML, prototype - 1 did not support the Dual Model. This prototype was developed using Smalltalk-80. Thus, the Dual Model schema was first converted to a simplified schema which could be compiled by VML. This implementation is discussed in [K90, B90]. In [D91] an enhanced version of this preliminary implementation is discussed. This implementation supported all the neccessary methods to perform the basic operations of this university OODB. Later in [P91a], a database browser was developed for this University database. In this implementation, the MVC classes supported by Smalltalk-80 were used to build the database browser. For performing basic operations, this implementation used methods written by [D91]. The following is an example of the class, 'Student' using VODAK/VML prototype - 1.

```
DEFINE VmlClass Student
   Category: 'UNIV-Db'
   instanceType:
     [subtypeof: PersonType;
    properties:
        [studentId: Integer;
        localAddress: AddressTypes;
        degree: String;
        lastEducation: String;
        organization: ^StudentOrganization;
        transcript: ^Transcript]].
```

An implementation of the university database using only C++ is discussed in [P91b]. The purpose of this implementation was to study the usability of the C++ language for development of complex, data_intensive applications. At that time, the VODAK/VML prototype - 2 was under development using C++. Thus, this implementation was also a preparation for work with VODAK/VML prototype-2.

The following is an example of the class 'Student', using only C++.

```
class Student : public Person {
friend  class Students;
private:
int                     Type;  //For Derived classes
char                    StudentId[MEDIUM_STRING_LENGTH];
Address                 LocalAddressObject;
char                    Degree[MEDIUM_STRING_LENGTH];
char                    LastEducation[LARGE_STRING_LENGTH];
Student*                NextStudent;
StudentOrganization*    Organization;
Transcript*             StudentTranscript;

public:
Student(void) {}
                    // constructor for Student
 ~Student(void) {}
                    // destructor for Student

                    // To Read and Write Student Id
   void  Rd_Student_StudentId(char*);
void  Wt_Student_StudentId(char*);

                    // To Read and Write Type
   void  Rd_Student_Type(int&);
   void  Wt_Student_Type(int&);

                    // To Read and Write Degree
   void  Rd_Student_Degree(char*);
   void  Wt_Student_Degree(char*);

                    // To Read and Write LastEducation
   void  Rd_Student_LastEducation(char*);
     void  Wt_Student_LastEducation(char*);

                    // To Read and Write Local Address
   void  Rd_Student_LocalAddress(AddressTuple*);
   void  Wt_Student_LocalAddress(AddressTuple*);

                    // To Assign Transcript to Student
      void  Assign_Student_Transcript(Transcript&);
```

```
                    // To Assign StudentOrganization
  void  Assign_Student_StudentOrganization(
StudentOrganization&);

                    // To Display Records
   void  Show_Student_Records(int, float, char*);

                    // To Print Complete Details of
                    // Student
   void  Pr_Full_Student(void);
   void  Pr_Half_Student(void);

void  Store_Object(void);
void  Read_Object(void);
void  Store_Deep_Object(void);
void  Read_Deep_Object(void);

   };
```

CHAPTER 3

GRAPHICAL SCHEMA OF UNIVERSITY OODB

As a first step of our thesis work, the university database schema was drawn using a graphical schema editor called OODINI. OODINI is a system for graphical schema representation developed at the New Jersey Institute of Technology [HGPN92] to represent object-oriented databases graphically.

The OODINI graphical language consists of labeled, directed graphs, where both vertices and edges are labeled. The vertex labels allow representation of class names and edge labels allow representation of relations and path methods.

3.1 Symbols Used by OODINI to Represent Database Elements

The following symbols are used by OODINI to represent the various elements in an object-oriented database Schema (Fig 3.1 shows the symbols).

- **Class**

  A rectangle (with name of the class printed inside).

- **Set class**

  A rectangle with double frame border.

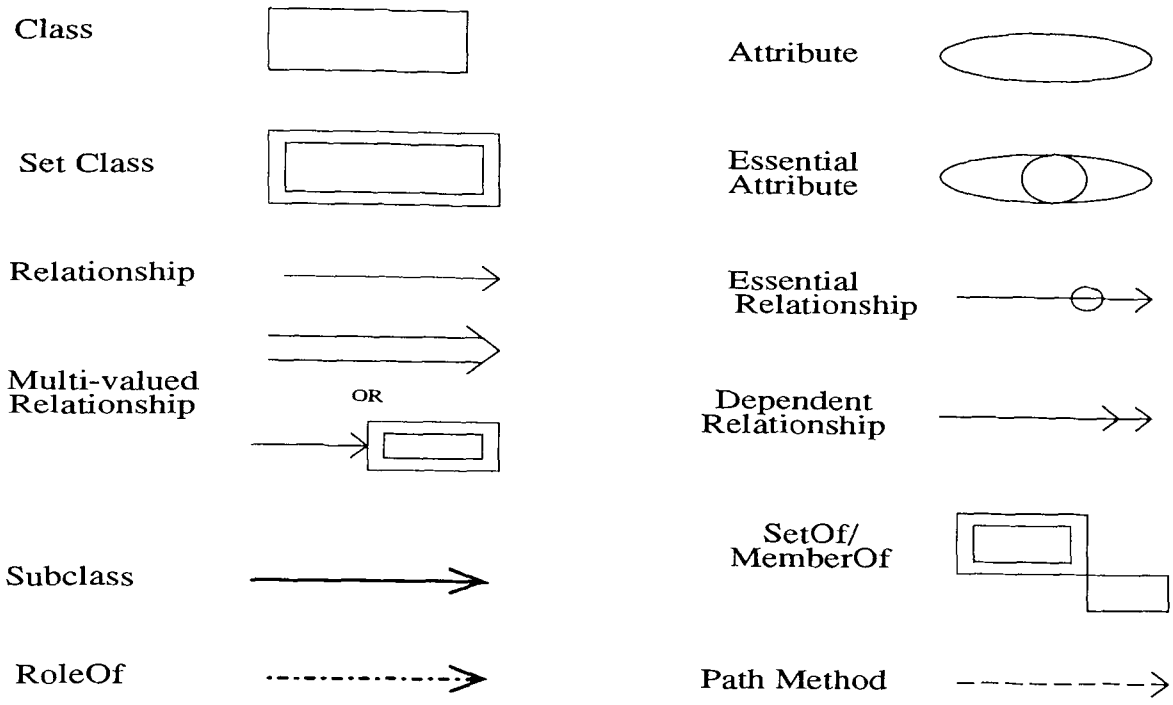

Figure 3.1 Symbols Used by OODINI

- **Attribute**

  An ellipse with name of attribute printed inside.

- **Essential attribute**

  Inscribed circle added to the ellipse representing an attribute.

- **Relationship**

  An arrow from one class to another.

- **Essential relationship**

  An arrow with a small circle behind the arrow head.

- **Dependent relationship**

A line with two arrow heads.

- **Multi-valued relationship**

  Either a dual-lined arrow, or a set class at the and of single-valued relationship.

- **Subtype (subclass)**

  A heavy line directed from the specialized class to the more general class.

- **Role**

  A directed, dashed-dotted, heavy line from the specialized class to the more general class.

- **Memberof/setof**

  The member class box and the set class box share exactly one corner.

- **Path method**

  A dashed, thin-lined arrow pointing from the class defining the method to the remote data items e.g., a class or an attribute. The elements on the path of the method may be hatched.

3.2 Options Available in OODINI for Building OODB Schema

OODINI is a menu-driven system. It provides menu options for the following operations.

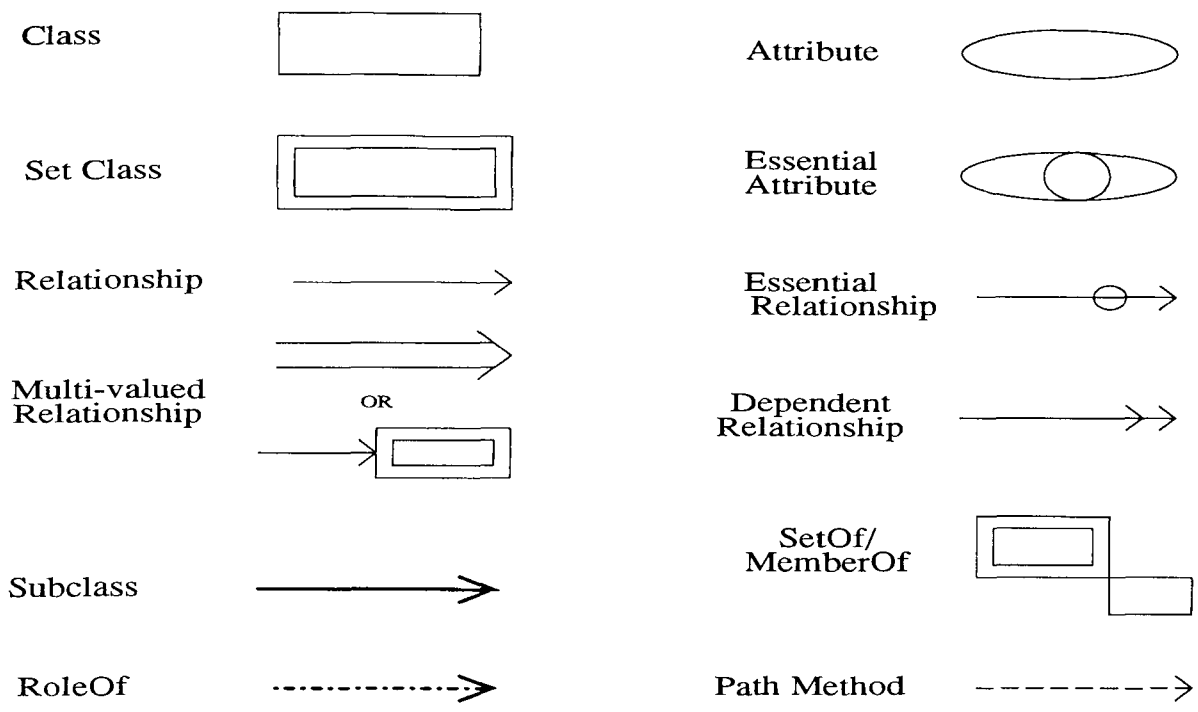

Figure 3.2 Symbols Used by OODINI

- **Input** - To draw an element
- **Move** - To move an element
- **Delete** - To delete an element

These operations can be applied to the following elements that can also be chosen from the menu.

- **Class** - Class
- **Relship** - User defined relationship
- **Attrib** - Attribute
- **Tuple** - A group of classes (Example: classes in a ternary relationship)

- **Set** - Set class
- **EssRel** - Essential Relationship
- **DepRel** - Dependent Relationship
- **MVRel** - Multi-Valued Relationship
- **MVDRel** - Multi-Valued Dependent Relationship
- **Roleof** - *Roleof* relation
- **Partof** - *Partof* Relation
- **Subcls** - *Categoryof* Relation

3.3 Building an OODB Schema Using OODINI

To draw a class, the class symbol and input option are selected from the menu, and the symbol is positioned at the desired location on the canvas. The name of the class is input from the keyboard as the system prompts for the name.

To add an attribute, essential attribute, or a set class to a given class, the appropriate symbol is selected from the menu and the class is selected in the schema already drawn. The name of the element is entered in response to the prompt. The mouse is used to position the element appropriately in relation to the class

User-defined relationships and generic relationships are added by first selecting the appropriate relationship type from the menu and then selecting the class from

which the relationship originates. The mouse button is held down after positioning the pointer on the originating class and dragged to the terminating class. Now, the desired relationship is drawn by the system.

To delete an element from the schema, select the element type and the Delete option from the menu, and then click the mouse button on the element. The element is deleted. The OODINI graphical elements incorporate constraints. For example, if a class is deleted, all the attributes associated with the class are also deleted.

To move an element, select the element type and the Move option from the menu, place the mouse pointer and drag the element to the desired location. If the class is moved, all the properties associated with the classes are also moved with it.

Viewing the Schema on the Screen

Roadmap: OODINI provides this option to view the complete schema at reduced size. This option can also be used to reposition the current working area on the canvas. To reposition the current working area, the 'focus rectangle' that appears on the road map is positioned over the desired portion of the schema. The area covered by the focus rectangle becomes the current working area visible on the main canvas. Horizontal and vertical scroll bars facilitate scrolling over the full schema to view and to edit it.

Obtaining Hardcopy of Graphical Schema

OODINI provides an option to convert, one screen at a time, the schema to a postscript file for obtaining hard copies of the schema. A facility to convert the

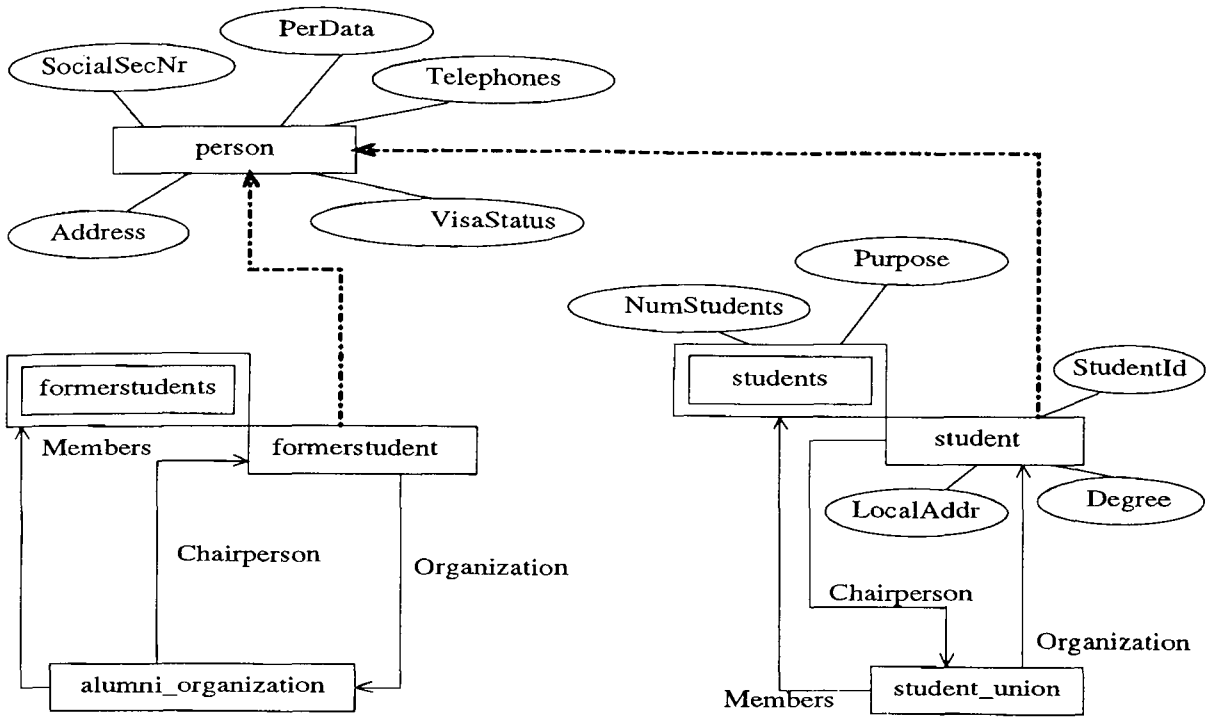

Figure 3.3 A Subschema of University Database

complete schema to a postscript file is not available at present. Fig 3.2 shows a portion of the university database schema represented graphically, using OODINI.

3.4 Generating Code from OODINI Graphical Schema

The OODINI system supports a code generator which generates OODB schema language code from the graphical representation. This is done in two steps. First, the graphical representation is converted into an abstract textual language called OODAL. Then, the schema in OODAL is converted to a specific OODB schema language. For more detailed information on code generation refer to [S92].

Currently, OODINI supports only conversion from OODAL to VML. This option

was used to convert the university database graphical schema into the VML schema. The following is a portion of the university database schema in VML, generated by OODINI.

```
SCHEMA .ooheap

OBJECTTYPE RoleSpec_InstType SUBTYPEOF Metaclass_InstType;

    PROPERTIES
            roleOf : {OID} ;

    METHODS
            defRoleClass(genClasses :  {OID});
            createRole(genInstances : {OID}) : OID;
            checkIsRoleOf(genClass :  OID) : BOOL;
            roleSpecializationOf() : {OID};

    IMPLEMENTATION
    METHODS

            defRoleClass(genClasses : {OID});
            {
            roleOf := genClasses;
            };

           createRole(genInstances : {OID}) : OID;
           {
            VAR roleObj : OID;
            VAR instance : OID;

            FORALL instance IN genInstances
            DO {
                IF SELF->checkIsRoleOf(instance->(OID)class()) == FALSE
                   THEN
                     RETURN NULL
                END
                }
            END;
            roleObj := SELF->(OID)new();
            roleObj->(OID)initRoleOf(genInstances);
```

```
            RETURN roleObj;
           };

           checkIsRoleOf(genClass : OID) : BOOL;
           {
            VAR x : OID;

            FORALL x IN roleOf
            DO {
                IF x == genClass
                  THEN
                     RETURN TRUE
                  END
                }
           END;
           RETURN FALSE;
          };

          roleSpecializationOf() : {OID};
          {
            RETURN roleOf;
          };

END;

OBJECTTYPE RoleSpec_InstInstType SUBTYPEOF Metaclass_InstInstType;
    PROPERTIES
           roleOf : {OID};

    METHODS
           initRoleOf (genInstances : {OID});
           roleSpecializationOf() : {OID};

    IMPLEMENTATION
    METHODS

           initRoleOf (genInstances : {OID});
           {
            roleOf := genInstances;
           };

           roleSpecializationOf() : {OID};
```

```
            {
             RETURN roleOf;
            };

            NOMETHOD
           {
            VAR x : OID;

            FORALL  x IN roleOf
             DO
             {
                RETURN(x.CurrentMethod(arguments));
             }
              END
            }
END;

CLASS ROLE_SPECIALIZATION_CLASS METACLASS Metaclass
    INSTTYPE RoleSpec_InstType
    INSTINSTTYPE RoleSpec_InstInstType
END;



OBJECTTYPE student_unionType [ student_unionClass : student_unionType,
           studentClass : studentType,
           studentsClass : studentsType ]
         SUBTYPEOF MetaClass_InstType;

     PROPERTIES
            Address : unknown_type;
            NumMembers : unknown_type;
            Members : studentsClass;
            Chairperson : studentClass;

     IMPLEMENTATION
      METHODS
            method_student_unionType() READONLY;
{
};
END;
```

```
OBJECTTYPE alumni_organizationType [ alumni_organizationClass :
  alumni_organizationType,
          formerstudentClass : formerstudentType,
          formerstudentsClass : formerstudentsType ]
     SUBTYPEOF MetaClass_InstType;

       PROPERTIES
           Members : formerstudentsClass;
           Chairperson : formerstudentClass;

       IMPLEMENTATION
        METHODS
     method_alumni_organizationType() READONLY;
     {
     };
END;

OBJECTTYPE formerstudentsType [ formerstudentsClass : formerstudentsType,
       formerstudentClass : formerstudentType ]
       SUBTYPEOF MetaClass_InstType;

    IMPLEMENTATION
     METHODS
    method_formerstudentsType() READONLY;
    {
    };
END;

OBJECTTYPE formerstudentType [ formerstudentClass : formerstudentType,
           formerstudentsClass : formerstudentsType,
           alumni_organizationClass : alumni_organizationType ]
           SUBTYPEOF MetaClass_InstType;

     PROPERTIES
            Organization : alumni_organizationClass;
            memberof : formerstudentsClass;

     IMPLEMENTATION
       METHODS
            method_formerstudentType() READONLY;
           {
           };
```

```
END;

OBJECTTYPE studentType [ studentClass : studentType,
            studentsClass : studentsType,
            student_unionClass : student_unionType ]
            SUBTYPEOF MetaClass_InstType;

       PROPERTIES
            LocalAddr : unknown_type;
            Degree : unknown_type;
            StudentId : unknown_type;
            Organization : student_unionClass;
            memberof : studentsClass;

       IMPLEMENTATION
       METHODS
            method_studentType() READONLY;
           {
           };
END;

OBJECTTYPE studentsType [ studentsClass : studentsType,
            studentClass : studentType ]
SUBTYPEOF MetaClass_InstType;

        PROPERTIES
            NumStudents : unknown_type;
            Purpose : unknown_type;
            setof : { studentClass };

    IMPLEMENTATION
      METHODS
            method_studentsType() READONLY;
            {
            };
END;

OBJECTTYPE personType [ personClass : personType ]
      SUBTYPEOF MetaClass_InstType;

      PROPERTIES
            VisaStatus : unknown_type;
```

```
            Telephones : unknown_type;
            PerData : unknown_type;
            SocialSecNr : unknown_type;
            Address : unknown_type;

      IMPLEMENTATION
        METHODS
            method_personType() READONLY;
            {
            };
END;

CLASS student_union
    INSTTYPE student_unionType [ student_union, student, students ]
END;

CLASS   alumni_organization
    INSTTYPE alumni_organizationType [ alumni_organization,
 formerstudent, formerstudents ]
END;

CLASS    formerstudents
    INSTTYPE formerstudentsType [ formerstudents, formerstudent ]
END;

CLASS   person
     INSTTYPE personType [ person ]
END;

CLASS   formerstudent METACLASS ROLE_SPECIALIZATION_CLASS
        INSTTYPE formerstudentType [ formerstudent, formerstudents,
alumni_organization ]
        INIT SELF->defRoleClass( { person } )
END;

CLASS   student METACLASS ROLE_SPECIALIZATION_CLASS
        INSTTYPE studentType [ student, students, student_union ]
        INIT SELF->defRoleClass( { person } )
END;

CLASSstudents
        INSTTYPE studentsType [ students, student ]
```

```
END;

END_SCHEMA;
```

The code produced by the OODINI code generator requires further changes to be made by a human. For instance, data types are not supplied by OODINI, and the code generator produces only "unknown-type." This has to be replaced by a correct data type.

CHAPTER 4

IMPLEMENTING UNIVERSITY OODB USING VODAK/VML

4.1 New VML Prototype

In our research we use VODAK/VML prototype – 2. This prototype has been developed at GMD-IPSI. Our experience during the development of the university database will also be used as a feedback for a future releases of VODAK/VML. As discussed earlier, this prototype – 2 supports separation of structural and semantic aspects in the class definition. Thus, we first define an object type. The object type studentType is defined below.

```
OBJECTTYPE studentType [ studentClass: studentType,
                         studentsClass: studentsType,
                         student_unionClass: student_unionType,
                         transcriptClass: transcriptType ,
                         personClass: personType]
                         SUBTYPEOF personType [personClass];

    INTERFACE
        PROPERTIES
            LastEducation: STRING;
            StudentId: INT;
            Degree: STRING;
            LocalAddr: STRING;
            Organization: student_unionClass;
            Transcript: transcriptClass;
            memberof_stu: studentsClass;
```

```
METHODS
    setLastEducation(lastedu: STRING) READONLY;
    getLastEducation(): STRING READONLY;
    setLocalAddr(locAddr: STRING) READONLY;
    getLocalAddr(): STRING READONLY;
    setOrganization(org: student_unionClass) READONLY;
    getOrganization(): student_unionClass READONLY;
    setTranscript(trans: transcriptClass) READONLY;
    getTranscript(): transcriptClass READONLY;
    setmemberof_stu(memof_stu: studentsClass) READONLY;
    getmemberof_stu(): studentsClass READONLY;
    Create_student (lastedu: STRING, loc_addr: STRING,
        org: student_unionClass, trans: TranscriptClass,
        memof: studentsClass, degr: STRING,
        stid: INT, addrs: STRING) READONLY;


IMPLEMENTATION
EXTERN prints(s: STRING);
EXTERN printi(i: INT);
EXTERN printr(v: REAL);
EXTERN endline();
METHODS

    setLastEducation(lastedu: STRING) READONLY;
    {LastEducation:= lastedu;};
    getLastEducation(): STRING READONLY;
    { RETURN LastEducation; };

    setLocalAddr(locAddr: STRING) READONLY;
    {LocalAddr:= locAddr;};
    getLocalAddr(): STRING READONLY;
    { RETURN LocalAddr; };

    setOrganization(org: student_unionClass) READONLY;
    {Organization:= org;};
    getOrganization(): student_unionClass READONLY;
    { RETURN Organization; };

    setTranscript(trans: transcriptClass) READONLY;
    {Transcript:= trans;};
    getTranscript(): transcriptClass READONLY;
```

```
{ RETURN Transcript; };

setmemberof_stu(memof_stu: studentsClass) READONLY;
{memberof_stu:= memof_stu;};
getmemberof_stu(): studentsClass READONLY;
{ RETURN memberof_stu; };

Create_student (lastedu: STRING, loc_addr: STRING,
     org: student_unionClass, trans: TranscriptClass,
     memof: studentsClass, degr: STRING,
     stid: INT, addrs: STRING) READONLY;
{

VAR p_person: personClass;
VAR s_student: studentClass;

p_person:= personClass->OID)new();
'SELF->initRoleof(p_person)'
s_student->setLastEducation(lastedu);
s_student->SYS_set_StudentId(stid);
s_student->setLocalAddr(loc_Addr);
s_student->SYS_set_Degree(degr);
s_student->setTranscript(trans_Addr);
s_student->SYS_set_Organization(org);
s_student->SYS_set_setmemberof_stu(memof);

s_student->setAddress(addrs);

};
END;
```

This object type student is *subtypeof* object type person. If SUBTYPEOF is not specified then the object type is a subtype of Metaclass_InstType. A Metaclass_Insttype is a built-in object type in VML. All the attributes are represented as properties. VML requires a keyword 'INTERFACE' preceding the keyword 'PROPERTIES' to specify the interface part of a class. VML supports primitive as well

as structured data types for attributes. The primitive data types are INT, REAL, BOOL and STRING. Four structured data types are defined below.

```
DATA TYPE StreetAddressType = [
            number: INT,
            street: STRING,
            unit: STRING] ;
DATA TYPE CityAddressType = [
            city: STRING,
            state: STRING,
            zip: STRING] ;
DATA TYPE AddressType = [
            streetaddress: sreetAddressType,
            cityaddress: CityAddressType] ;
DATA TYPE CompanyAddressType = [
            department: DepartmentType,
            streetaddress: StreetAddressType,
            cityaddress: CityAddressType] ;
```

Now we will discuss how to define relationships. Relationships are specified as properties in VML. For each relationship one needs to pass a formal class parameter. For example, TranscriptClass is a formal class parameter for the relationship *Transcript* to the class **student**. Similarly, Organization is a formal parameter for the relationship *Membership*.

In the Dual Model one object type can be shared by many classes. This can be realized by using formal class parameters. The line 'transcriptClass: transcriptType' specifies that the transcriptclass can be a class which has an object type transcript. Similarly, 'organizationClass: organization' specifies that **organization** is a class which has an object type organization.

For each attribute, VML supports two built-in methods, to read and to write its value. The read method is described as SYS_<prop> (): <typeofprop> and the write method is described as SYS_set_<prop>(param: <typeofprop>). Here 'prop' is a selector for the property and 'typeof Prop' is a data type. These methods are generated by the VML compiler. For example, for the attribute *StudentId* the read methods is 'SYS_StudentId(): INT'; and the write method is 'SYS_set_StudentId(param: INT)';

Such methods are also defined for complex attributes. In addition, one can write such methods as follows.

setStudentId(stid: INT) READONLY;

getStudentId(): INT READONLY;

The methods that we have written contain the keyword 'READONLY'. This keyword shows that the method is side effect free. In a class definition, VML allows external methods written in C++. For example, prints, printi and endline shown in the implementation part of the class definition, are such methods.

For each object type we write a method to initialize instances of the class which has this object type. For example, for the class **student** we have a write method *Create_student.* This method accepts all the attribute values and instances for relationships as parameters.

The class **student** is *roleof* the class **person**, if we want to we create an instance of the class **person** 'p_person', then we use

p_person:= personClass->OID)new();

to link an existing student with the roleof relation we use

'SELF->initRoleof(p_person)' Then we assign all the values, which are passed as parameters, using write methods. The class **student** is defined below:

```
CLASS student METACLASS ROLE_SPECIALIZATION_CLASS
INSTTYPE studentType [ student, students, student_union,
        transcript, person]
INIT student->defRoleClass( person )
END;
```

In VML the generic specialization relationships *roleof* and *categoryof* are defined by a metaclass, i.e., a metaclass was written to support the behaviour of such specialization relationships.

The first line of the above class definition specifies that the class **student** is *roleof* the class **person**. This is specified by the INIT clause. The keyword INSTTYPE, given after the keyword CLASS, specifies that the studentType is the object type of the class **student**. If a METACLASS is not specified then each class is an instance of a built–in class VMLCLASS. The INIT clause specifies that **student** is *roleof* the class **person**. Other classes can be similarly written in VML.

4.2 Compilation and Execution of the Schema

The compilation of the schema consists of the following steps.

a. Compilation of the Schema.

b. Compilation of the CreateDatabase file

c. Compilation of the Application

Compilation of the Schema:

1. vc -dc <schemaname>: This command compiles the VML file and converts it into a C++ file.

2. tc <schemaname>: This command compiles the C++ file of the schema.

Compilation of the CreateDatabase file: This file creates the database

1. vc -c crdb

2. tc crdb.o

   The file crdb contains the code to create a database for the schema. It contains the name of the schema <schemaname>, and a set of commands for creating the database.

Compilation of the Application: The <app> file is an application written in VML

1. vc -c <app>

2. tc <app.>o

3. vodak crdb <schemaname>: Using VODAK, crdb and the schema are used to create a database.

4. vodak <app> <schemaname>: This command is used to run the program using the application and the schema.

4.3 Precompilation Modifications Required

As discussed in Chapter 3, the schema is prepared using OODINI. Then, using the code generator, the VML code for the university database is generated.
There are a number of changes to be made to the schema, code due to the following reasons.

1. The code generator of OODINI was developed before the release of VODAK-VML prototype – 2. Thus, it used earlier and now outdated specifications.

2. OODINI was developed using a general OODB model rather than the Dual Model. Thus, the code for classes which share an object type cannot be generated.

   Thus, the following changes had to be made:

1. Changing the Data type Schema: A part of the schema code that is not automatically generated deals with the data types. We had to adapt a previous manually written data type schema to the changed university database.

2. Editing

   a. The object types and classes had to be put in the same order as in the earlier prototype of the university database, to achieve more clarity.

   b. For every object type we had to add the heading 'INTERFACE' to the interface part of the object type definition of VML.

c. We had to change every ';' as separator in data types and formal parameters to ','.

In the example in Section 4.1 'INTERFACE' was not generated and therefore not in the code. In the example of data types every ',' was a ';' in the generated code and we replaced them.

3. Specifying the subtypes of object types: The SUBTYPEOF relation of object types had to be specified as shown in the same example of Section 4.1.

4. Checking for the matching of the properties in the supertype-subtype classes (since overriding is not supported by VML): The property names had to be checked such that the same selector does not appear in two classes, e.g., the attribute *address* may not occur in the class **person** and the class **student**.

5. Fixing class parameters and adding class parameters which are missing. The class parameters should be in the same order as in the object type definition and we had to make sure that there are no missing parameters. These order differences are due to the fact that the graphical representation does not define an order.

4.4 Additional Changes Necessitated by Limitations of VML

1. Small sample schemas with primitive data types, set data types, complex data

types and object types were designed. Complex data types had to be replaced by primitive data types.

2. Testing for multiple inheritance: Multiple inheritance had to be eliminated.

3. Testing of *roleof* and *categoryof* relationships: As already mentioned, in the Dual Model representation the *roleof* and *categoryof* relations are semantic relations. We checked using the university database if these functionalities are supported by the VML system. In the university database we used *categoryof* relationships because the VML system supported *categoryof* relationship only. but now the *roleof* relationship is also supported.

4. Using Read and Write system defined functions and methods: Read and write system defined functions are methods to read and write data. They had to be written for complex data types.

The following steps were to be performed before compiling a schema or creating a database to clear up temporary structures left over from a previous compiler run:

1. ipcs: This prints out information about shared memory and semaphores that are currently active in the system.

   ipcrm -s <sem_id>: This removes the semaphore set.

   ipcrm -m <shm_id>: This removes the shared memory ID.

2. tc -M <schemaname>: Generates a file <schemaname>.MK, which is similar to a makefile. This file is useful, when we call external methods written in

C++. Suppose, all the external C++ functions are written in the file IO.C. Then we add IO.o to <schemaname>.MK so that, it compiles IO.C and links it to the VODAK module. In addition, we can add external libraries to this file. For example X libraries to develop the graphical interface to the university database.

4.5 Features Not Supported by VML

During the tests of the VML system using the university database it was observed that the following features are currently not supported by the VML System:

1. Multiple inheritance is not supported: One class cannot have two superclasses.

2. Overriding of properties is not supported: A subclass cannot have a property selector used in any of its superclasses.

3. The following data types are not supported: a. nested record type, b. dictionary. The data types "dictionary" and "nested record type" described in the VML design specification document [KNBD92] were not supported by the VML compiler.

4. Reader methods and writer methods for any of the structured data types are not supported. They are supported only for the primitive data types INT, STRING, REAL, and BOOL.

5. A superclass should be defined as

   <superclass> METACLASS GenCatSpecClass

   otherwise the statement

   <subclass> METACLASS CatSpecClass may not work.

6. Currently VML can compile only between 25 and 40 classes, depending upon the class sizes.

CHAPTER 5

GRAPHICAL USER INTERFACE FOR UNIVERSITY OODB

As mentioned earlier, we developed a database browser for the university OODB. The current version of VML allows us to write application programs. Only a simple, menu-driven user interface can be written using VML or C++ to use the database. But there is no built-in support for writing a graphical user interface. We, therefore, decided to develop a graphical user interface in order to support database browsing.

5.1 What Constitutes the Interface?

We have chosen X-Windows, Motif and C++ to develop our interface. Note that VODAK/VML prototype-2 has been developed using C++. Thus, use of C++ for the user interface facilitates easy integration of the interface with the Vodak/VML OODBMS.

The X-Window system is a network-based windowing system. The basic windowing functions are implemented in a library called 'Xlib'. To facilitate development of graphical user interfaces, a toolkit known as Xt-Intrinsics has been developed by MIT. This toolkit provides the basic functions for creation/destruction of widgets, setting resources for widgets, etc. For example, for creating and managing a wid-

get, Xt provides a simple function, which is XtCreateManagedWidget(widget-name, widget-class, parent, resource-array, num-of-resources). There is no such simple way to create and manage a widget using Xlib. A series of function calls with long lists of parameters are needed to do this job using Xlib. A primitive widget set, called Athena widget set, is provided with Xt by MIT. For rapid development of sophisticated graphical user interfaces, a number of commercial widget sets are available. Two such toolkit sets are very popular and are widely used for development of GUI's: The OSF/Motif toolkit, developed by The Open Software Foundation, and the Open-Look Widget Set, developed by AT&T. We have chosen to use OSF/Motif Widget Set version 1.0, available on our system.

Incidentally, Motif widgets have also been developed using an object-oriented architecture. Each widget is an instance of a widget class, which inherits the properties of its base classes. Thus, there is a parent-child relationship among the Motif widgets. A subtree of widgets can be created to form a component of an application interface. In our interface system, each subtree of widgets is created and managed by a C++ class. There are a total of 36 C++ classes in the system. Out of these 36 classes, 16 classes constitute a reusable library. The major functions of the reusable library are as follows:

1. **To create Motif Menu Window :**

   The Motif Menu Window is a widget that can contain, as children, a Menu Bar at the top of the window and a work area below the Menu Bar.

2. **To create Motif Menu Bar:**

   The Motif Menu Bar is a widget that can contain Pulldown and Cascade menu widgets.

3. **To Create Motif Pulldown Menu:**

   The Motif Pulldown Menu is a widget that can be pulled down by a mouse click. This widget will normally have PushButtons corresponding to the menu options.

4. **To Create Cascade Buttons:**

   The Motif Cascade Button forms the connection between the main option on the Menu Bar and the sub-options of the Pulldown Menu. The name of the Cascade button is the name of the main menu option.

5. **To create PushButtons:**

   As already discussed, Push Buttons correspond to the menu options.

6. **To provide an abstract base class for the command objects:**

   We have implemented a class for each of the menu options in order to associate the option buttons with the actions. The actions associated with each menu option are encapsulated in a C++ class. A single instance, called command object, is created to handle the specific actions associated with the corresponding option. The common functionalities and member variables required by all such command classes are implemented in a base class in the reusable library.

7. **To provide methods for creation, management and destruction of widgets:**

   The common functions such as creation of widgets, destruction of widgets and setting the default resources for each widget are encapsulated in a C++ class in the reusable library. Each component class for creating a widget sub-tree can be derived from this base class to use the above functionalities.

8. **To provide system startup functions:**

   Functions for connecting to the X-server, creating the top level window for the application, realizing the top level widget, entering into the application event loop, and storage of common data, such as pointer to the 'Display' and pointer to the structure of the application context are encapsulated in a single C++ class in the reusable library. An instance of this class is created at application startup time to invoke the above functions.

A more detailed discussion of the various Motif widgets can be found in [O91]. The library can be used, without modification, by other applications that may be developed in the future. The remaining classes in the interface implement the database browser based on the Model-View-Controller principle, to be discussed in the next section.

5.2 Architecture of GUI

The underlying architecture used in the GUI is the Model-View-Controller architecture, supported by the Smalltalk environment [GR83]. This is a powerful approach for presenting multiple views in an interactive application. Therefore, it can be effectively used to implement a database browser, which is concerned with presenting multiple views of a database. The functions necessary to handle database updates can be easily integrated into this scheme.

In the Model-View-Controller architecture, the Model represents the data or picture to be manipulated for viewing. View is a specific view of the data or picture that is presented to the user. The Controller interacts with the user and coordinates with Model to retrieve the desired portion of the data or picture to be displayed by View.

C++, unlike Smalltalk, does not have a built-in facility to implement the MVC architecture. However, recently, efforts have been made to develop MVC using C++. One such example is discussed in [DY92] using C++ and OSF/Motif. Definition and implementation of some base classes are also given in the above reference. In our implementation of MVC for the university database, we have used the ideas given in the above reference, but chosen to define and implement our own classes. This was done to customize MVC to University database browser. In our implementation, the task of integration with the DBMS can be accomplished by modifying only two of the 36 classes.

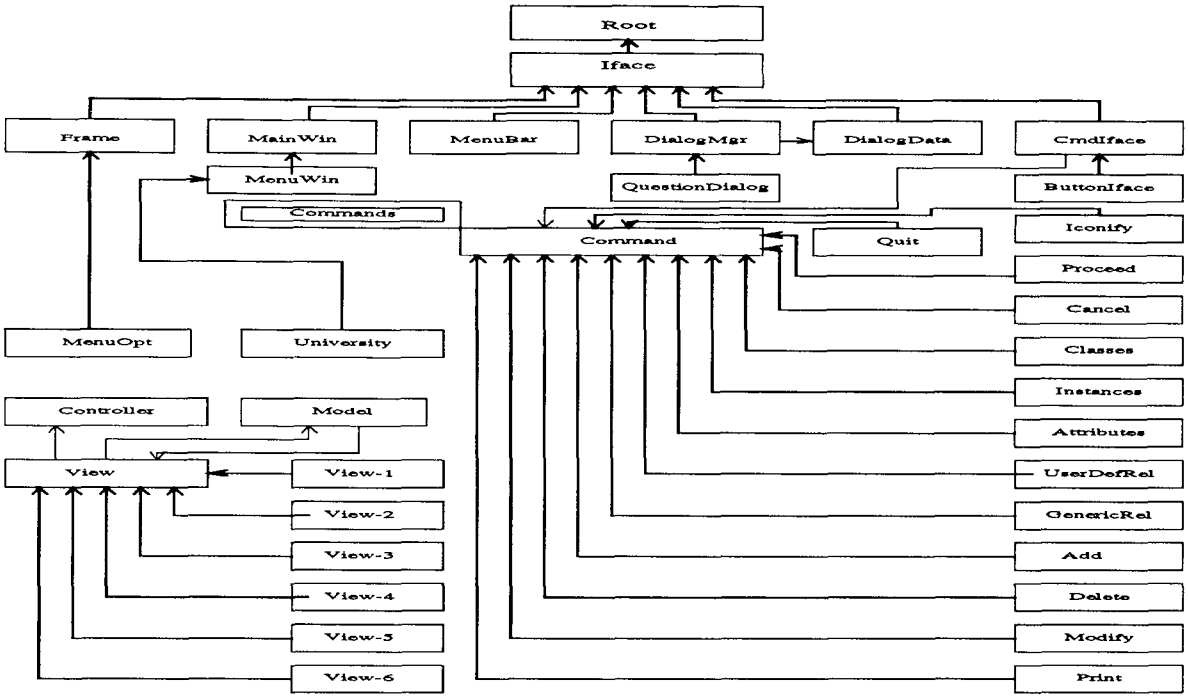

Figure 5.1 Schematic Diagram of C++ Classes Used in the GUI

5.3 Description of the C++ Classes used for University Database Browser

Fig 5.1 shows a schematic diagram of the C++ classes used in the interface.

5.3.1 Reusable Library Classes

The name of the reusable library is called libUniv.a. The following is a description of the main classes in this library.

1. **Root:** This is an abstract base class from which all component classes are derived. This class handles creation of widgets, destruction of widgets and stores the name of each subtree of widgets.

2. **Iface:** This is also an abstract base class, derived from Root and specialized to handle default resource setting and error handling.

3. **Frame:** This class implements methods to make connection to an X-server, initialize the Xt-Intrinsics, realize the top-level widget of the interface, store a pointer to the 'Display' variable of the X-Server and enter the application's event loop. An instance of this class is created by the system at startup time. The component classes check for the existence of this object before creating any widget, in order to avoid a system crash.

4. **Command:** This class is a base class from which all the command classes corresponding to the menu options are derived. The common functions associated with menu options are implemented in this class.

5. **MainWin:** This class is used for creating the Main Window of the interface to contain the Menu Bar and Work Area for the Application.

6. **MenuBar:** This class is used for creating the Menu Bar for the interface. The menu bar will contain Pulldown and Cascade Menu widgets of the interface.

7. **ButtonIface:** This class is used for creating instances of Push Button widgets for the interface.

8. **DialogMgr:** This class is used to create a base widget for dialog boxes. Classes used for displaying error messages can be derived from this class.

9. **QuestionDialog:** This class displays a question dialog box to get a confirmation from the user. The type of question depends on the requirements. This class is derived from DialogMgr.

10. **QuitWin:** This class implements the functionalities needed for quitting the application. It displays a question dialog box to get the user's confirmation before quitting.

11. **CmdIface:** This class has been created to provide a clear separation between the Motif widgets and the functions associated with database browsing. Such a separation helps to modify the way a database browser works without interfering with the X-Window/Motif part of the software. The functionalities needed to relate a menu option to a command object are implemented by the class 'CmdIface'.

5.3.2 MVC Classes

As mentioned earlier, each menu option in this implementation is represented by a command object. An instance of each of the MVC classes is created by the system at startup time. The menu options are classified into groups based on their purpose. In this section, we discuss the classes corresponding to each menu option. The layout of the menu options on the screen and the operations from the user point of view are shown in Figure 5.4 (Screen Layout of database browser).

The functions of each class are discussed below.

Menu Classes

Classes for SELECTION group menu options:

1. **Classes** - Selects all classes in the database schema
2. **Instances** - Selects the instances of a given class
3. **Attributes** - Selects the attributes of a given object type
4. **UserDefRel** - Selects the user defined relationships for a given object
5. **GenericRel** - Selects the generic relations, i.e., roleof, subtypeof, categoryof or setof.

The above five classes form the sub-options under the main option called **SELECTION** in the main menu.

Classes for ACTION group menu options:

1. **Add** - To add an object of a given type to the database
2. **Print** - To list the items chosen from the SELECTION menu
3. **Edit** - To modify the values of the properties
4. **Delete** - To delete a given object from the database

The above four options form the sub-options under the main option called **ACTION** in the main menu.

Classes for CONFIRM group menu options:

1. **Proceed** - To proceed with the indicated action

2. **Cancel** - To cancel the selected action

   The above two options form the sub-options under the main option called **CONFIRM** in the main menu.

Core Classes of M-V-C

A block diagram showing the classes representing the Model, the Views and the Controller of the interface to the university database is shown in Fig 5.2. The diagram also shows relationships between the Model, the View and the Controller and the relationship between the Model and DBMS. An arrow represents communication between the connected classes in the direction of the arrow head. Two way communication is represented by two directed lines with opposing directions between the two classes.

1. **Controller**

   The Controller class implements the Controller part of MVC. When a menu option is selected, the corresponding command object reports to the Controller class. Also, whenever an item is selected from a list window, it is reported to the Controller class. The Controller decides on the action to be taken and informs the Model of the data to be retrieved from the database or update to be made to it.

2. **Model**

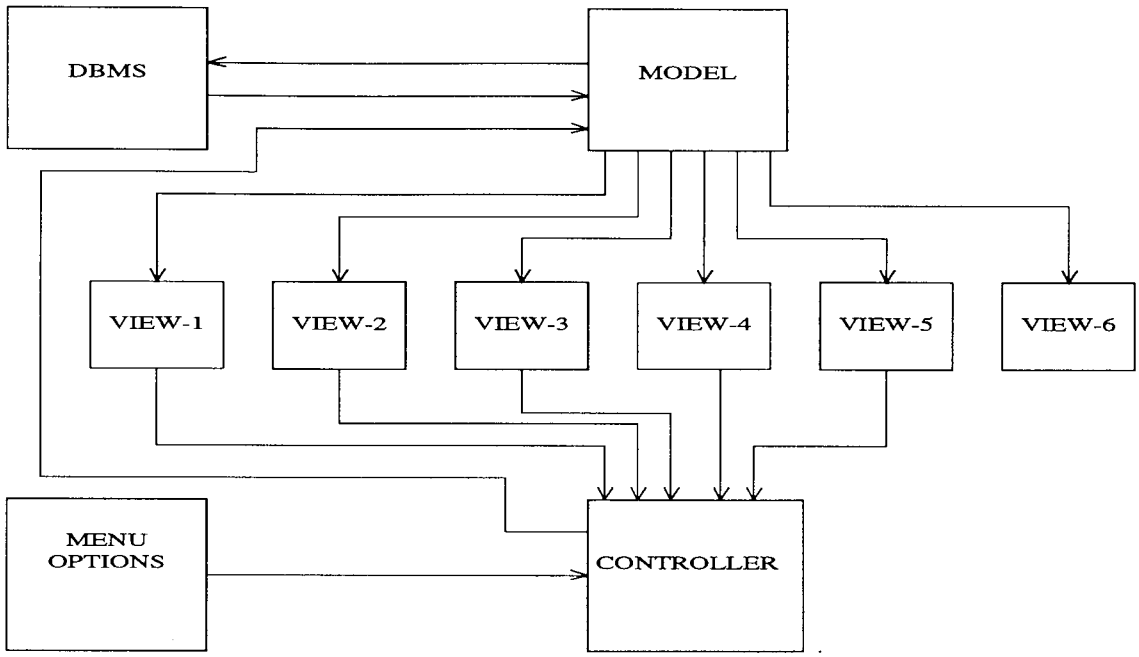

Figure 5.2 Block Diagram of Model-View-Controller

The Model class implements the Model part of MVC. The Model class object accepts requests from Controller and interacts with the OODBMS to get the data to be displayed or the updates to be performed. If the request is for fetching data from the database, the Model object stores the fetched data in its character string arrays and informs the appropriate View object that the data is ready for display. If the request is a database update, the Model object calls the appropriate method in the application program of the database and hands over the data to the application program for updating the database.

Classes View1, View2, View3, View4, View5 and View6 implement the View part of the MVC.

3. **View1**

   The View1 class implements a Motif Scrolled List window called Object Window for displaying the list of objects. Once the database object list to be displayed is fetched by the Model object, the View1 object accesses the string array of the Model object in which the list is stored. Then the View1 object does the conversion needed for displaying and then displays the list.

4. **View2**

   The View2 class implements a Motif Scrolled List called Property Window. The View2 object is similar to the View1 object, except that it displays the property names in the Property Window.

5. **View3**

   The View3 class implements a Motif Scrolled List called Value Window. The View3 object is also similar to the View1 object, except that it displays the property values in the Value Window

6. **View4**

   The View4 class implements a Motif Scrolled Text Window called Edit Window. The View4 class implements the functionalities needed for editing property values of an object.

7. **View5**

The View5 class implements a Motif Scrolled Text Window called Query Window. This class implements the functionalities to input and execute query statements.

8. **View6**

   The View6 class implements a Motif 'read-only' Scrolled Text Window called Query Result Window. The View6 object accesses the string array of the Model object, where the list of classes is stored, and displays the list in Query Result Window.

   The View classes are declared as friends of the Model class. This feature in C++ allows the friend classes to access the private data members of the class in which such a declaration is done. Thus, the View objects are able to retrieve the data stored in the character string arrays of the Model object for displaying them on the screen.

5.4 Screen Layout of the Browser

The screen layout for this database browser is shown in Figs. 5.3 to 5.5. Fig. 5.3 shows the various windows and the purpose of each window.View1 is displayed in Window-1, etc. There are six windows, four on the upper half and two on the lower half. The first three windows - Object Window, Property Window and Value Window are List Windows used for displaying the instances, property names and properties

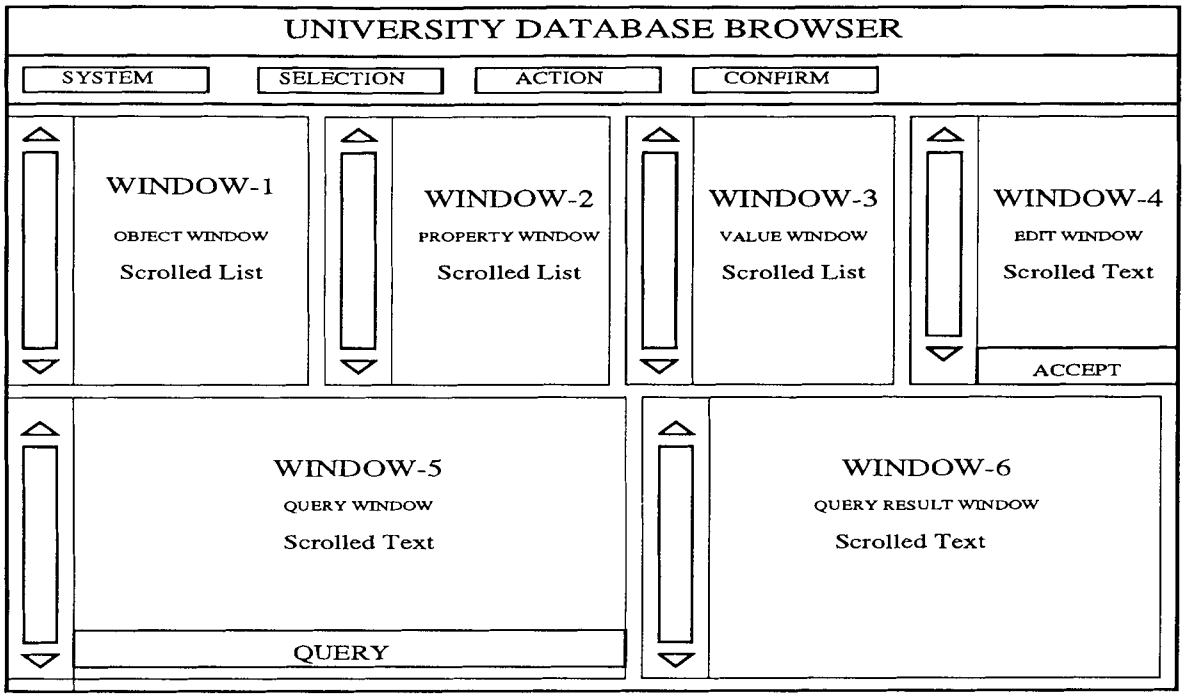

Figure 5.3 Database Browser Screen Showing the Windows

values respectively. The fourth window, called Edit Widow, is a Text Window which is used for editing.

The bottom left Window, called Query Window, is a Text Window. This is used for writing and executing query statements. The bottom right window, called Query Result Window, is also a Text Window but 'Read Only'. This window is used for displaying the list of classes and the query results.

Fig. 5.4 shows the menu options of all menus in the database browser.

Fig. 5.5 shows the windows with a display of sample data. A list of objects is displayed in the Object Window. When the object 'Person-P1' is selected from Object Window, option 'Attributes' is selected from the 'SELECTION' menu-pane and 'Print' is selected from the 'ACTION' menu-pane, the property names of the

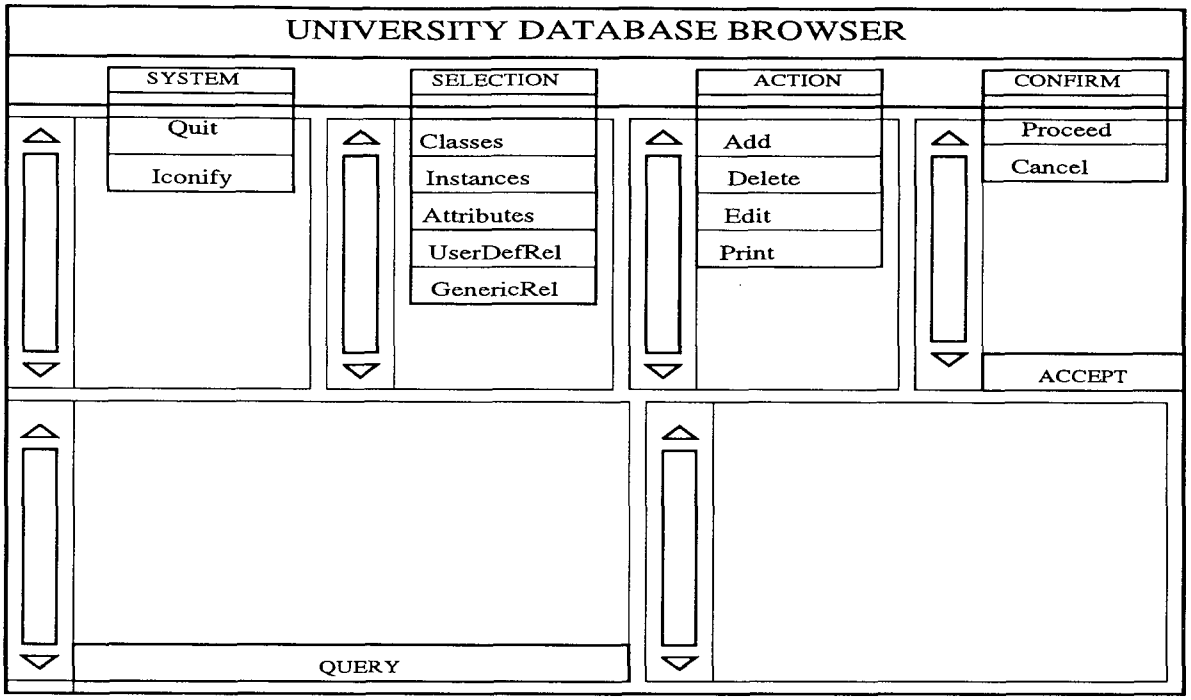

Figure 5.4 Database Browser Screen Showing the Menu Options

object 'Person-P1' are displayed on the Property Window. Now, the property name, 'Name' is selected from Property Window and the option 'Print' is selected from the 'ACTION' menu-pane so that the name of object, 'Person-P1' is displayed in the Value Window. If the name is to be modified, the name 'Simon Lam' is selected and the 'Edit' ACTION is selected. Then the name appears in the Edit Window. The name can now be modified. After modification, the 'Accept' button has to be pressed in order to update the database.

Window-6 in Fig 5.5 shows a display of class names from the university database.

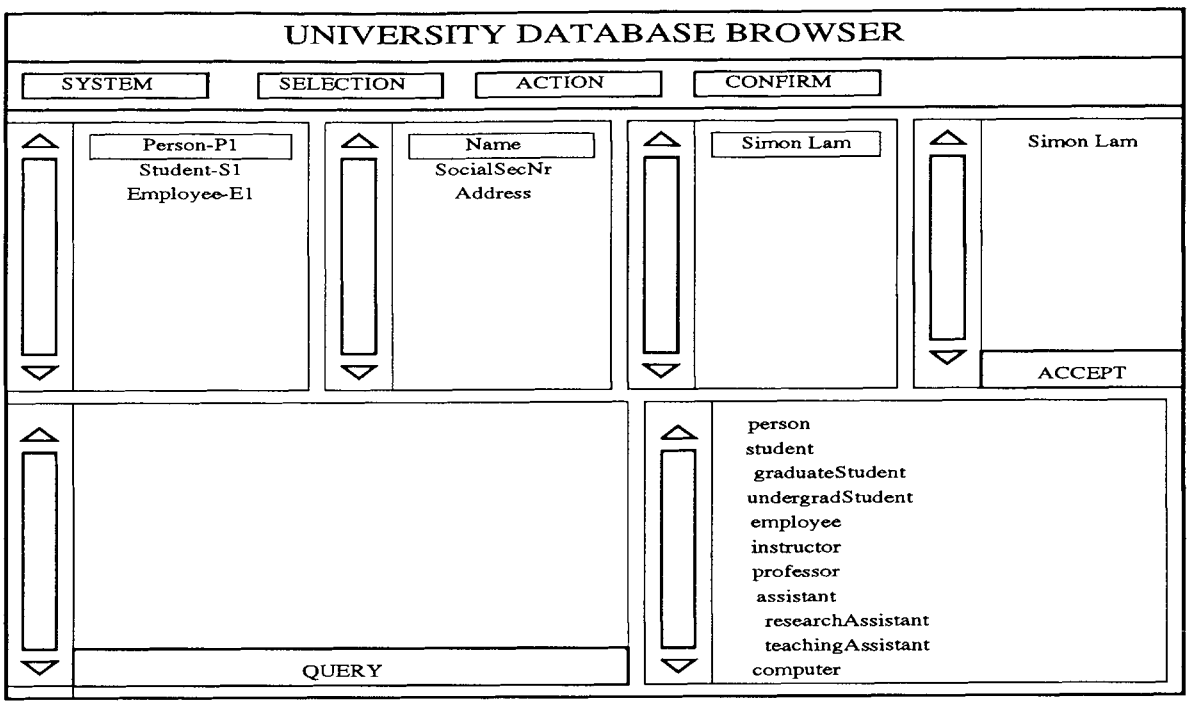

Figure 5.5 Database Browser Screen with Sample Data

5.5 Integration of Database Browser with the Database

As mentioned earlier, the functions for integration of the database browser with the database are confined to only two classes, thus, highly simplifying this task. These classes are the Model class and the Controller class. This section specifies what code is to be inserted and where, in order to integrate the interface with the DBMS.

Controller Class

The code for decision making has to be inserted into the method 'Proceed' in the Controller class. This method is invoked when the 'Proceed' button is pressed. The following member variables are used for decision making. Which variables are to be used depends upon the type of ACTION desired by the user.

1. _action - stores the action desired by the user.

2. _selection - stores the selection made by the user.

3. _className - stores the name of the class whose objects are needed.

4. _instanceName - stores the object name whose properties are needed.

5. _propName - stores the name of a property whose value is needed.

6. _propValue - stores the value of a property, which is to be modified.

7. _queryStatement - stores the query statement entered in Query Window.

8. _win - stores the List Window name from which an item was selected. This is used to identify the type of item selected.

When a query statement is entered and the 'Query' button is pressed, the query statement is simply passed to the Model class. In this case the 'Proceed' method will not be invoked. The code necessary for this purpose is already available in the Controller class.

Model Class

The Model class must have the appropriate code for interacting with the database. The code for interaction with the DBMS is inserted into the following methods of Model class, depending upon the nature of interaction needed.

1. getClasses() - To get the list of classes in the schema.

2. getInstances() - To get the list of instances of a given class.

3. getAttributes() - To get the list of attribute names for a given class.

4. getUserRel() - To get user defined relationships for a given class.

5. getGenRel() - To get generic relationships for a given class.

6. getPropNames() - To get the values of a given property name.

7. queryDBMS() - To send a query statement to DBMS.

8. updateDBMS() - To update information in the database.

When the return value of a function call is to be passed on to a List Window, the value has to be stored in the member variable '_outList.' If the return value is to be passed on to a Text Window, then the value has to be stored in the member variable '_outText.' Similarly, the input data from a List Window, passed from the Controller object, will be available in the member variable '_inList' and the input data from a Text Window will be available in the member variable '_inText.' '

CHAPTER 6

CONCLUSIONS

In this thesis we have implemented a university environment database using the VODAK/VML prototype – 1. VODAK/VML is the first OODB which allows separation of structural and semantic aspects of a class definition. Similarly, the university database that we have developed is the first implemented database which uses this separation.

Initially, we investigated all the features supported by VODAK/VML. Then, based on the features which are currently available, we developed the university database. We have reported features of VODAK/VML which are currently not available to GMD–IPSI, so that they can incorporate them into the next version. We also developed a database browser for this university database using C++/MOTIF.

APPENDIX A

UNIVERSITY OODB SCHEMA CLASSES

```
SCHEMA twentyschema

IMPORT Datatypes FROM Datatypes;
IMPORT CatSpecSchema FROM CatSpecSchema;
IMPORT SemanticSchema FROM SemanticSchema;

DEFINE aspect 1

// 1. class person

  OBJECTTYPE personType [ personClass : personType ]
                        SUBTYPEOF Metaclass_InstType;

      INTERFACE
        PROPERTIES
                Address : STRING;
                SocialSecNr : INT;
                PerData : STRING;
                VisaStatus : VisaType;
                Telephones : TelephonesType;
        METHODS
        setAddress(anAddr: STRING) READONLY;
        getAddress() : STRING READONLY;
        setPerData(Pdata: STRING) READONLY;
        getPerData() : STRING READONLY;
        setVisaStatus(vstat: VisaType) READONLY;
        getVisaStatus() : STRING READONLY;
        setTelephones(aset : TelephonesType) READONLY;
        getTelephones() : TelephonesType READONLY;
        create_person(visa:STRING,adr:AddressType,ssn:INT,
          pername:STRING)  READONLY;

        IMPLEMENTATION
            EXTERN prints(s:STRING);
        EXTERN printi(i:INT);
```

```
EXTERN printr(v:REAL);
EXTERN reads():STRING;
EXTERN readi():INT;
EXTERN endline();
METHODS
setAddress(anAddr: STRING) READONLY;
{Address := anAddr;};

getAddress() : STRING READONLY;
{ RETURN Address; };

setTelephones(aset :{ TelephoneType}) READONLY;
//assigning a set of telephones
{Telephones := aset;};

getTelephones() : {TelephoneType} READONLY;
{RETURN Telephones;};

setPerData(Pdata: STRING) READONLY;
{PerData := Pdata;};

getPerData() : STRING READONLY;
{ RETURN PerData; };

setVisaStatus(vstat: VisaType) READONLY;
{VisaStatus := vstat;};

getVisaStatus() : STRING READONLY;
{ RETURN VisaStatus; };

create_person(visa:STRING,adr:AddressType,
 ssn:INT,pername:STRING)  READONLY;
{
        VAR address        : AddressType;
        VAR Perdata       : STRING;
        VAR tele_ph        : TelephoneType;
        VAR tele_phs       : {TelephoneType};

SELF->setAddress(adr);
SELF->SYS_set_SocialSecNr(ssn);
SELF->SYS_set_VisaStatus(visa);
SELF->SYS_set_PerData(pername);
```

```
        prints('enter person telephoneNumber number? ');
        tele_ph:= readi ();
        INSERT tele_ph INTO tele_phs;
        SELF->setTelephones(tele_phs);
        endline(); prints('create person is complete');
endline();
        };

        print_person() READONLY;
        {
                VAR addr : AddressType;
                VAR visa     : VisaType;
                VAR name       : STRING;
                VAR ssn         : INT;
                VAR tele_ph        : TelephoneType;
                VAR tele_phs       : TelephonesType;

        addr := SELF->getAddress();
        prints('address is    :'); prints(addr); endline();

        ssn := SELF->SYS_SocialSecNr();
        prints('ssn is    :'); printi(SocialSecNr); endline();

        name := SELF->SYS_PerData();
        prints('name is     :'); prints(PerData); endline();

        visa := SELF->SYS_VisaStatus();
        prints('Visastatus is    :'); prints(visa); endline();

        FORALL (tele_ph IN tele_phs)
        {
        //tele_ph := SELF->SYS_Telephone();
        printi(tele_ph); endline();
        };

        };

END;

CLASS   person

        INSTTYPE personType [person]
```

```
END;

// 2.  class student

OBJECTTYPE studentType [ studentClass : studentType,
                         studentsClass : studentsType,
                         student_unionClass : student_unionType,
                         transcriptClass : transcriptType ,
                         personClass : personType]
                         SUBTYPEOF personType [personClass];

      INTERFACE
        PROPERTIES
                LastEducation : STRING;
                StudentId : INT;
                Degree : STRING;
                LocalAddr : STRING;
                Organization : student_unionClass;
                Transcript : transcriptClass;
                memberof_stu : studentsClass;
        METHODS
                setLastEducation(lastedu: STRING) READONLY;
                getLastEducation()  : STRING READONLY;
                setLocalAddr(locAddr: STRING) READONLY;
                getLocalAddr()  : STRING READONLY;
                setOrganization(org: student_unionClass) READONLY;
                getOrganization(): student_unionClass READONLY;
                setTranscript(trans: transcriptClass) READONLY;
                getTranscript(): transcriptClass READONLY;
                setmemberof_stu(memof_stu : studentsClass) READONLY;
                getmemberof_stu(): studentsClass READONLY;
                Create_student (lastedu :STRING,loc_addr:STRING,
                degr:STRING,stid :INT,trans:transcriptClass,
                addrs:STRING) READONLY;


        IMPLEMENTATION
        EXTERN prints(s:STRING);
        EXTERN printi(i:INT);
        EXTERN printr(v:REAL);
        EXTERN endline();
```

```
METHODS
setLastEducation(lastedu: STRING) READONLY;
{LastEducation := lastedu;};
getLastEducation()  : STRING READONLY;
{ RETURN LastEducation; };

setLocalAddr(locAddr: STRING) READONLY;
{LocalAddr := locAddr;};
getLocalAddr()  : STRING READONLY;
{ RETURN LocalAddr; };

setOrganization(org : student_unionClass) READONLY;
{Organization := org;};
getOrganization(): student_unionClass READONLY;
{ RETURN Organization; };

setTranscript(trans : transcriptClass) READONLY;
{Transcript := trans;};
getTranscript(): transcriptClass READONLY;
{ RETURN Transcript; };

setmemberof_stu(memof_stu : studentsClass) READONLY;
{memberof_stu := memof_stu;};
getmemberof_stu(): studentsClass READONLY;
{ RETURN memberof_stu; };

Create_student (lastedu :STRING,loc_addr:STRING,
degr:STRING,stid :INT,trans:transcriptClass,
addrs:STRING) READONLY;

{
VAR addr       : STRING;
VAR addr2      : STRING;
VAR sid        : INT;
VAR locAddr    : STRING;
VAR deg        : STRING;
VAR last_edu   : STRING;
VAR p_person   : personClass;
VAR s_student  : studentClass;

addr := addrs;
sid  := stid;
```

```
        last_edu := lastedu;
        locAddr := loc_addr;
        deg := degr;

        SELF->setLastEducation(lastedu);
        SELF->SYS_set_StudentId(stid);
        SELF->setLocalAddr(loc_addr);
        SELF->SYS_set_Degree(degr);
        SELF->setTranscript(trans);

        SELF->setAddress(addrs);
        addr2 := SELF->getAddress(); endline();
        prints('person address from student '); prints(addr2);
endline();

        };
END;

CLASS   student   METACLASS CATEGORY_SPECIALIZATION_CLASS

        INSTTYPE studentType [ student, students, student_union,
                               transcript, person]

        INIT    student->defCategory( person ,aspect)

END;

// 3. Class students

OBJECTTYPE studentsType [ studentsClass : studentsType,
                          studentClass : studentType ]
                                SUBTYPEOF Metaclass_InstType;

      INTERFACE
        PROPERTIES
                NumOfStudents : INT;
                Pupose : STRING;
                setof : studentClass;
        METHODS
setsetof(stof : studentClass) READONLY;
getsetof(): studentClass READONLY;
```

```
IMPLEMENTATION
        EXTERN prints(s:STRING);
        EXTERN printi(i:INT);
        EXTERN printr(v:REAL);
        EXTERN endline();
METHODS
        setsetof(stof : studentClass) READONLY;
        {setof := stof;};
        getsetof(): studentClass READONLY;
        { RETURN setof; };
END;

CLASS   students
INSTTYPE studentsType [ students, student ]
END;

// 4. class employee

OBJECTTYPE employeeType [ employeesClass : employeesType,
employeeClass : employeeType,
                        resumeClass : resumeType,
                        personClass : personType ]
                                SUBTYPEOF personType [personClass];

      INTERFACE
        PROPERTIES
                EmployeeNr : INT;
                TotWeekHours : INT;
                OfficeAddr : STRING;
                Started : INT;
                SalaryRate : REAL;
                PhoneNumber : TelephoneType;
                Type : STRING;
                Resume : resumeClass;
                Supervisor_emp : employeeClass;
                Supervisees : employeesClass;
                memberof_emps : employeesClass;
        METHODS
                setEmployeeNr(empno : INT) READONLY;
                getEmployeeNr(): INT READONLY;
                setOfficeAddr(offaddr : STRING) READONLY;
                getOfficeAddr(): STRING READONLY;
```

```
            setStarted(start : INT) READONLY;
            getStarted(): INT READONLY;
            setSalaryRate(salrate : REAL) READONLY;
            getSalaryRate(): REAL READONLY;
            setPhoneNumber(phone : TelephoneType) READONLY;
            getPhoneNumber(): TelephoneType READONLY;
            setResume(res : resumeClass) READONLY;
            getResume(): resumeClass READONLY;
            setSupervisor_emp(super_emp : employeeClass) READONLY;
            getSupervisor_emp(): employeeClass READONLY;
            setSupervisees(super_ses : employeesClass) READONLY;
            getSupervisees(): employeesClass READONLY;
            setmemberof_emps(memof_emps : employeesClass) READONLY;
            getmemberof_emps(): employeesClass READONLY;

            create_emp(empnr:INT,totwkhr:INT,offaddr: STRING,
            start:INT,Sal:REAL,Phone:TelephoneType,
            res:resumeClass )READONLY;

    IMPLEMENTATION
EXTERN printi(i:INT);
        EXTERN prints(s:STRING);
    EXTERN printr(v:REAL);
    EXTERN endline();

    METHODS
    setEmployeeNr(empno : INT) READONLY;
    {EmployeeNr := empno;};
    getEmployeeNr(): INT READONLY;
    { RETURN EmployeeNr; };

    setOfficeAddr(offaddr : STRING) READONLY;
    {OfficeAddr := offaddr;};
    getOfficeAddr(): STRING READONLY;
    { RETURN OfficeAddr; };

    setStarted(start : INT) READONLY;
    {Started := start;};
    getStarted(): INT READONLY;
    { RETURN Started; };

    setPhoneNumber(phone : TelephoneType) READONLY;
```

```
{PhoneNumber := phone;};
getPhoneNumber(): TelephoneType READONLY;
{ RETURN PhoneNumber;};

setSalaryRate(salrate : REAL) READONLY;
{ SalaryRate := salrate;};
getSalaryRate(): REAL READONLY;
{ RETURN SalaryRate; };

setType(type : STRING) READONLY;
{Type := type;};
getType(): STRING READONLY;
{ RETURN Type; };

setResume(res : resumeClass) READONLY;
{Resume := res;};
getResume(): resumeClass READONLY;
{ RETURN Resume; };

setSupervisor_emp(super_emp : employeeClass) READONLY;
{Supervisor_emp := super_emp;};
getSupervisor_emp(): employeeClass READONLY;
{ RETURN Supervisor_emp; };

setSupervisees(super_ses : employeesClass) READONLY;
{Supervisees := super_ses;};
getSupervisees(): employeesClass READONLY;
{ RETURN Supervisees; };

setmemberof_emps(memof_emps : employeesClass) READONLY;
{memberof_emps := memof_emps;};
getmemberof_emps(): employeesClass READONLY;
{ RETURN memberof_emps; };

create_emp(empnr:INT,totwkhr:INT,offaddr: STRING,start:INT,
 Sal:REAL,Phone:TelephoneType,res:resumeClass )READONLY;
{

VAR addrs       : STRING;
VAR addr2      : STRING;
```

```
                    SELF->setEmployeeNr(empnr);
                    SELF->setSalaryRate(Sal);
                    SELF->setPhoneNumber(Phone);
                    SELF->SYS_set_TotWeekHours(totwkhr);
                    SELF->setOfficeAddr(offaddr);
                    SELF->setResume(res);

                    SELF->setAddress(addrs);
                    addr2 := SELF->getAddress();
                    endline();
                };
END;


CLASS   employee        METACLASS CATEGORY_SPECIALIZATION_CLASS

        INSTTYPE employeeType [ employees, employee,
                                        resume, person ]


        INIT    employee->defCategory( person,aspect )

END;

// 5. class employeesType

OBJECTTYPE employeesType [ employeesClass : employeesType,
                           employeeClass : employeeType ]
                                SUBTYPEOF Metaclass_InstType;

      INTERFACE
        PROPERTIES
                Purpose : STRING;
                NumOfEmployees : INT;
                setof : employeeClass;
        METHODS
                setsetof(stof : employeeClass) READONLY;
                getsetof(): employeeClass READONLY;

        IMPLEMENTATION
        EXTERN prints(s:STRING);
        EXTERN printi(i:INT);
```

```
        EXTERN printr(v:REAL);
        EXTERN endline();
        METHODS
        setsetof(stof : employeeClass) READONLY;
        {setof := stof;};
        getsetof(): employeeClass READONLY;
        { RETURN setof; };
END;


CLASS   employees

        INSTTYPE employeesType [ employees, employee ]

END;

// 6. class transcript

OBJECTTYPE transcriptType [ transcriptClass : transcriptType,
                            course_recordsClass : course_recordsType,
                            sectionsClass : sectionsType,
                            studentClass : studentType ]
                                SUBTYPEOF Metaclass_InstType;

      INTERFACE
        PROPERTIES
                Date : DateType;
                Student : studentClass;
                CurrentSections : sectionsClass;
                CourseRecords : course_recordsClass;
        METHODS
                setDate(date : DateType) READONLY;
                getDate(): DateType READONLY;
                setStudent(stud : studentClass) READONLY;
                getStudent(): studentClass READONLY;
                setCurrentSections(cursec : sectionsClass) READONLY;
                getCurrentSections(): sectionsClass READONLY;
                setCourseRecords(currec : course_recordsClass) READONLY;
                getCourseRecords(): course_recordsClass READONLY;
                create_trans(dat:DateType) READONLY;

         IMPLEMENTATION
```

```
        EXTERN prints(s:STRING);
        EXTERN printi(i:INT);
        EXTERN printr(v:REAL);
        EXTERN endline();
        METHODS
        setDate(date : DateType) READONLY;
        {Date := date;};
        getDate(): DateType READONLY;
        { RETURN Date; };

        setStudent(stud : studentClass) READONLY;
        {Student := stud;};
        getStudent(): studentClass READONLY;
        { RETURN Student; };

        setCurrentSections(cursec : sectionsClass) READONLY;
        {CurrentSections := cursec;};
        getCurrentSections(): sectionsClass READONLY;
        { RETURN CurrentSections; };

        setCourseRecords(courrec : course_recordsClass) READONLY;
        {CourseRecords := courrec;};
        getCourseRecords(): course_recordsClass READONLY;
        { RETURN CourseRecords; };


        create_trans(dat:DateType) READONLY;
        {
        SELF->SYS_set_Date(dat);
        };
END;

CLASS   transcript

        INSTTYPE transcriptType [ transcript, course_records,
                                            sections, student]

END;

// 7.  class student_union

OBJECTTYPE student_unionType [student_unionClass : student_unionType,
```

```
                                studentsClass : studentsType,
                                studentClass : studentType,
                                employeesClass : employeesType ]
                            SUBTYPEOF Metaclass_InstType;

      INTERFACE
        PROPERTIES
                Address : STRING;
                NumMembers : INT;
                Chairperson : studentClass;
                Members : studentsClass;
                Workers : employeesClass;
METHODS
        setAddress(addr: STRING) READONLY;
        getAddress(): STRING READONLY;
        setChairperson(chper : studentClass) READONLY;
        getChairperson(): studentClass READONLY;
                setMembers(membrs : studentsClass) READONLY;
                getMembers(): studentsClass READONLY;
                setWorkers(wkers : employeesClass) READONLY;
                getWorkers(): employeesClass READONLY;
                create_studunion(addrs:STRING,nmem:INT,
 workrs:employeesClass) READONLY;

        IMPLEMENTATION
        EXTERN prints(s:STRING);
        EXTERN printi(i:INT);
        EXTERN printr(v:REAL);
        EXTERN endline();
        METHODS
        setAddress(addr : STRING) READONLY;
        {Address := addr;};
        getAddress(): STRING READONLY;
        { RETURN Address; };

        setChairperson(chper : studentClass) READONLY;
        {Chairperson := chper;};
        getChairperson(): studentClass READONLY;
        { RETURN Chairperson; };

        setMembers(membrs : studentsClass) READONLY;
        {Members := membrs;};
```

```
        getMembers(): studentsClass READONLY;
        { RETURN Members; };

        setWorkers(wkers : employeesClass) READONLY;
        {Workers := wkers ;};
        getWorkers(): employeesClass READONLY;
        { RETURN Workers; };

        create_studunion(addrs:STRING,nmem:INT,workrs:employeesClass)
  READONLY;
        {
           SELF->setAddress(addrs);
           SELF->setWorkers(workrs);
        };
END;


CLASS   student_union

        INSTTYPE student_unionType [ student_union, students,
                                                  student, employees ]

END;

// 8. class sections

OBJECTTYPE sectionType [ sectionClass : sectionType,
                         sectionsClass : sectionsType,
                         studentsClass : studentsType,
                         crsectionsClass : crsectionsType,
                         instructorClass : instructorType ]
                                 SUBTYPEOF Metaclass_InstType;

      INTERFACE
        PROPERTIES
                SectionNo : INT;
                Capacity : INT;
                Time : TimeType;
                Room : RoomType;
                NoOfStudents : INT;
                Instructor : instructorClass;
                Sections : crsectionsClass;
```

```
        Students : studentsClass;
        memberof_secs : sectionsClass;
        memberof_crse : crsectionsClass;
METHODS
        setTime(time : TimeType) READONLY;
        getTime(): TimeType READONLY;
        setRoom(rm : RoomType) READONLY;
        getRoom(): RoomType READONLY;
        setInstructor(inst : instructorClass) READONLY;
        getInstructor(): instructorClass READONLY;
        setSections(sect : crsectionsClass) READONLY;
        getSections(): crsectionsClass READONLY;
        setStudents(studs : studentsClass) READONLY;
        getStudents(): studentsClass READONLY;
        setmemberof_secs(memof_sec : sectionsClass) READONLY;
        getmemberof_secs(): sectionsClass READONLY;
        setmemberof_crse(memof_crse : crsectionsClass) READONLY;
        getmemberof_crse(): crsectionsClass READONLY;
        create_secs(time:TimeType,room:RoomType,
        inst:instructorClass) READONLY;

IMPLEMENTATION
EXTERN prints(s:STRING);
EXTERN printi(i:INT);
EXTERN printr(v:REAL);
EXTERN endline();
METHODS
setTime(time : TimeType) READONLY;
{Time := time ;};
getTime(): TimeType READONLY;
{ RETURN Time; };

setRoom(rm : RoomType) READONLY;
{Room := rm;};
getRoom(): RoomType READONLY;
{ RETURN Room; };

setInstructor(inst : instructorClass) READONLY;
{Instructor := inst;};
getInstructor(): instructorClass READONLY;
{ RETURN Instructor; };
```

```
        setSections(sect : crsectionsClass) READONLY;
        {Sections := sect;};
        getSections(): crsectionsClass READONLY;
        { RETURN Sections; };

        setStudents(studs : studentsClass) READONLY;
        {Students := studs;};
        getStudents(): studentsClass READONLY;
        { RETURN Students; };

        setmemberof_secs(memof_sec : sectionsClass) READONLY;
        {memberof_secs := memof_sec;};
        getmemberof_secs(): sectionsClass READONLY;
        { RETURN memberof_secs; };

        setmemberof_crse(memof_crse : crsectionsClass) READONLY;
        {memberof_crse := memof_crse;};
        getmemberof_crse(): crsectionsClass READONLY;
        { RETURN memberof_crse; };

        create_secs(time:TimeType,room:RoomType,inst:instructorClass)
  READONLY;
        {
                    SELF->setTime(time);
                    SELF->setRoom(room);
                    SELF->setInstructor(inst);
        };

END;


CLASS   section

        INSTTYPE sectionType [ section, sections, students,
                                    crsections, instructor ]

END;

// 9. class sections

OBJECTTYPE sectionsType [ sectionsClass : sectionsType,
                          sectionClass : sectionType ]
```

```
                                SUBTYPEOF Metaclass_InstType;

      INTERFACE
        PROPERTIES
                Purpose : STRING;
                NumOfSections : INT;

        IMPLEMENTATION
        EXTERN prints(s:STRING);
        EXTERN printi(i:INT);
        EXTERN printr(v:REAL);
        EXTERN endline();
END;


CLASS   sections

        INSTTYPE sectionsType [ sections, section ]

END;

// 10. class crsections

OBJECTTYPE crsectionsType [ crsectionsClass : crsectionsType,
                            sectionClass : sectionType,
                            courseClass : courseType ]
                                SUBTYPEOF Metaclass_InstType;

      INTERFACE
        PROPERTIES
                NumSections : INT;
                Course : courseClass;
                setof :  sectionClass ;
        METHODS
                setCourse(crse : courseClass) READONLY;
                getCourse() : courseClass READONLY;
                setsetof(stof :  sectionClass ) READONLY;
                getsetof() :  sectionClass  READONLY;
        create_crsec(secnum:INT,cour:courseClass) READONLY;

        IMPLEMENTATION
        EXTERN prints(s:STRING);
```

```
        EXTERN printi(i:INT);
        EXTERN printr(v:REAL);
        EXTERN endline();
        METHODS

        setCourse(crse : courseClass) READONLY;
        {Course := crse;};
        getCourse() : courseClass READONLY;
        { RETURN Course; };

        setsetof(stof :  sectionClass ) READONLY;
        {setof := stof;};
        getsetof() :  sectionClass  READONLY;
        { RETURN setof; };
        create_crsec(secnum:INT,cour:courseClass) READONLY;
        {
                    SELF->SYS_set_NumSections(secnum);
                    SELF->setCourse(cour);
};
END;


CLASS   crsections

        INSTTYPE crsectionsType [ crsections, section, course ]

END;

// 11. class course

OBJECTTYPE courseType [ courseClass : courseType,
                        coursesClass : coursesType,
                        crsectionsClass : crsectionsType ]
                                SUBTYPEOF Metaclass_InstType;

      INTERFACE
        PROPERTIES
                Name_course : STRING;
                Department : STRING;
                CreditHour : INT;
                Number : INT;
                Sections : crsectionsClass;
```

```
        Prereq : coursesClass;
        memberof_cous : coursesClass;
METHODS
        setSections(secs : crsectionsClass) READONLY;
        getSections(): crsectionsClass READONLY;
        setPrereq(prereq : coursesClass) READONLY;
        getPrereq(): coursesClass READONLY;
        setmemberof_cous(memof_cous : coursesClass) READONLY;
        getmemberof_cous(): coursesClass READONLY;
        create_course(courname:STRING,dept:STRING,crdhr:INT,
        numb:INT,secs:crsectionsClass) READONLY;

IMPLEMENTATION
EXTERN prints(s:STRING);
EXTERN printi(i:INT);
EXTERN printr(v:REAL);
EXTERN endline();
METHODS
setSections(secs : crsectionsClass) READONLY;
{Sections := secs;};
getSections(): crsectionsClass READONLY;
{ RETURN Sections; };

setPrereq(prereq : coursesClass) READONLY;
{Prereq := prereq ;};
getPrereq(): coursesClass READONLY;
{ RETURN Prereq; };

setmemberof_cous(memof_cous : coursesClass) READONLY;
{memberof_cous := memof_cous;};
getmemberof_cous(): coursesClass READONLY;
{ RETURN memberof_cous; };

create_course(courname:STRING,dept:STRING,crdhr:INT,numb:INT,
secs:crsectionsClass) READONLY;
{
            SELF->SYS_set_Name_course(courname);
            SELF->SYS_set_CreditHour(crdhr);
            SELF->SYS_set_Number(numb);
            SELF->SYS_set_Department(dept);
            SELF->setSections(secs);
```

```
};
END;

CLASS   course

        INSTTYPE courseType [ course, courses, crsections ]

END;


// 12. class courses


OBJECTTYPE coursesType [ coursesClass : coursesType,
                         courseClass : courseType ]
                                 SUBTYPEOF Metaclass_InstType;

      INTERFACE
        PROPERTIES
                Purpose : STRING;
                NumOfCourses : INT;

        IMPLEMENTATION
        EXTERN prints(s:STRING);
        EXTERN printi(i:INT);
        EXTERN printr(v:REAL);
        EXTERN endline();
END;


CLASS   courses

        INSTTYPE coursesType [ courses, course ]

END;

// 13. class instructor

OBJECTTYPE instructorType [ instructorClass : instructorType,
                            instructorsClass : instructorsType,
                            sectionsClass : sectionsType,
                            employeesClass : employeesType,
```

```
                    employeeClass : employeeType,
                    resumeClass : resumeType,
                    personClass : personType]
                SUBTYPEOF employeeType [ employeesClass,
                                         employeeClass,
                                         resumeClass,
                                         personClass ];

INTERFACE
  PROPERTIES
          TeachEval : REAL;
          Sections : sectionsClass;
          memberof_ins: instructorsClass;
  METHODS
          setTeachEval(teacheval : REAL) READONLY;
          getTeachEval(): REAL READONLY;
          setSections(secs : sectionsClass) READONLY;
          getSections(): sectionsClass READONLY;
          setmemberof_ins(memof_ins: instructorsClass) READONLY;
          getmemberof_ins(): instructorsClass READONLY;
  Create_instructor (secs:sectionsClass,techeval:REAL) READONLY;

  IMPLEMENTATION
  EXTERN prints(s:STRING);
  EXTERN printi(i:INT);
  EXTERN printr(v:REAL);
  EXTERN endline();
  METHODS
  setTeachEval(teacheval : REAL) READONLY;
  {TeachEval := teacheval;};
  getTeachEval(): REAL READONLY;
  { RETURN TeachEval; };

  setSections(secs : sectionsClass) READONLY;
  {Sections := secs;};
  getSections(): sectionsClass READONLY;
  { RETURN Sections; };

  setmemberof_ins(memof_ins: instructorsClass) READONLY;
  {memberof_ins := memof_ins;};
  getmemberof_ins(): instructorsClass READONLY;
  { RETURN memberof_ins; };
```

```
        Create_instructor (secs:sectionsClass,techeval:REAL) READONLY;
{
                    VAR empno2      : INT;
                    VAR purpose  : STRING;

                    SELF->setSections(secs);
                    SELF->setTeachEval(techeval);
                    //SELF->setPurpose(purpose);
                    empno2 := SELF->getEmployeeNr();
                    endline();

                };
END;


CLASS   instructor      METACLASS CATEGORY_SPECIALIZATION_CLASS

        INSTTYPE instructorType [ instructor, instructors, sections,
                    employees, employee, resume, person ]


        INIT    instructor->defCategory( employee, aspect )

END;

// 14. class resume

OBJECTTYPE resumeType [ resumeClass : resumeType,
                        employeesClass : employeesType,
                        employeeClass : employeeType,
                        personClass : personType]

                                SUBTYPEOF Metaclass_InstType;

      INTERFACE
        PROPERTIES
                JobTitle : STRING;
                PersonalData : personClass;
        METHODS
                setPersonalData(persdata : personClass) READONLY;
```

```
                getPersonalData(): personClass READONLY;
create_res(job:STRING,perdat:personClass) READONLY;

        IMPLEMENTATION
        EXTERN prints(s:STRING);
        EXTERN printi(i:INT);
        EXTERN printr(v:REAL);
        EXTERN endline();
        METHODS
        setPersonalData(persdata : personClass) READONLY;
        {PersonalData := persdata;};
        getPersonalData(): personClass READONLY;
        { RETURN PersonalData; };
create_res(job:STRING,perdat:personClass) READONLY;
{
                    SELF->SYS_set_JobTitle(job);
                    SELF->setPersonalData(perdat);
};
END;


CLASS   resume

        INSTTYPE resumeType [ resume, employees, employee, person ]

END;

// 15. class course_record

OBJECTTYPE course_recordType [ course_recordClass : course_recordType,
                               course_recordsClass : course_recordsType,
                               courseClass : courseType ]
                                SUBTYPEOF Metaclass_InstType;

      INTERFACE
        PROPERTIES
                SemesterTaken : STRING;
                Grade : REAL;
                Course : courseClass;
                memberof_cou : course_recordsClass;
        METHODS
                setGrade(grade : REAL) READONLY;
```

```
                getGrade(): REAL READONLY;
                setCourse(crse : courseClass) READONLY;
                getCourse(): courseClass READONLY;
                setmemberof_cou(memof_cou : course_recordsClass) READONLY;
                getmemberof_cou(): course_recordsClass READONLY;

        IMPLEMENTATION
        EXTERN prints(s:STRING);
        EXTERN printi(i:INT);
        EXTERN printr(v:REAL);
        EXTERN endline();
        METHODS

        setGrade(grade : REAL) READONLY;
        {Grade := grade;};
        getGrade(): REAL READONLY;
        { RETURN Grade; };

        setCourse(crse : courseClass) READONLY;
        {Course := crse;};
        getCourse(): courseClass READONLY;
        { RETURN Course; };

        setmemberof_cou(memof_cou : course_recordsClass) READONLY;
        {memberof_cou := memof_cou;};
        getmemberof_cou(): course_recordsClass READONLY;
        { RETURN memberof_cou; };

END;



CLASS   course_record

        INSTTYPE course_recordType [ course_record, course_records, course ]

END;

// 16. class course_records

OBJECTTYPE course_recordsType [ course_recordsClass : course_recordsType,
                                course_recordClass : course_recordType,
```

```
                                courseClass : courseType,
                                transcriptClass : transcriptType ]
                                SUBTYPEOF Metaclass_InstType;

        INTERFACE
          PROPERTIES
                  Purpose : STRING;
                  NumCourseRecords : INT;
                  Transcript : transcriptClass;
                  setof : course_recordClass;
          METHODS
                  setTranscript(trans : transcriptClass) READONLY;
                  getTranscript(): transcriptClass READONLY;
                  setsetof(stof : course_recordClass) READONLY;
                  getsetof(): course_recordClass READONLY;

          IMPLEMENTATION
          EXTERN prints(s:STRING);
          EXTERN printi(i:INT);
          EXTERN printr(v:REAL);
          EXTERN endline();
          METHODS
          setTranscript(trans : transcriptClass) READONLY;
          {Transcript := trans;};
          getTranscript(): transcriptClass READONLY;
          { RETURN Transcript; };

          setsetof(stof : course_recordClass) READONLY;
          {setof := stof;};
          getsetof(): course_recordClass READONLY;
          { RETURN setof; };

END;


CLASS   course_records

        INSTTYPE course_recordsType [ course_records, course_record,
                                              course, transcript ]

END;
```

```
// 17. class formerstudent

CLASS   formerstudent   METACLASS CATEGORY_SPECIALIZATION_CLASS

        INSTTYPE studentType [ formerstudent, formerstudents,
                               alumni_organization, transcript, person ]


        INIT    formerstudent->defCategory( person,aspect )

END;

// 18. class formerstudents

CLASS   formerstudents

        INSTTYPE studentsType [ formerstudents, formerstudent ]

END;


// 19. class alumni_organization

CLASS   alumni_organization

        INSTTYPE student_unionType [ alumni_organization, formerstudents,
                                     formerstudent, employees ]

END;

// 20. class undergrad

OBJECTTYPE undergradType [ undergradClass : undergradType,
                           faculty_memberClass : faculty_memberType,
                           studentClass : studentType,
                           studentsClass : studentsType,
                           student_unionClass : student_unionType,
                           transcriptClass : transcriptType,
                           personClass : personType]
                       SUBTYPEOF studentType [ studentClass,
                                               studentsClass,
                                               student_unionClass,
```

```
                                        transcriptClass,
                                        personClass];

INTERFACE
  PROPERTIES
          Project : STRING;
          Minor : MajMinType;
          Major : MajMinType;
          Year : YearType;
          Supervisor_facm : faculty_memberClass;
  METHODS
          setProject(proj : STRING) READONLY;
          getProject(): STRING READONLY;
          setMinor(min : MajMinType) READONLY;
          getMinor(): MajMinType READONLY;
          setMajor(maj : MajMinType) READONLY;
          getMajor(): MajMinType READONLY;
          setYear(year : YearType) READONLY;
          getYear(): YearType READONLY;
          setSupervisor_facm(superfac : faculty_memberClass) READONLY;
          getSupervisor_facm(): faculty_memberClass READONLY;
          create_undergrad(facmem:faculty_memberClass,
         Proj:STRING,majr:MajMinType) READONLY;

  IMPLEMENTATION
  EXTERN prints(s:STRING);
  EXTERN printi(i:INT);
  EXTERN printr(v:REAL);
  EXTERN endline();
  METHODS
  setProject(proj : STRING) READONLY;
  {Project := proj;};
  getProject(): STRING READONLY;
  {RETURN Project;};

  setMinor(min : MajMinType) READONLY;
  {Minor := min;};
  getMinor(): MajMinType READONLY;
  { RETURN Minor; };

  setMajor(maj : MajMinType) READONLY;
  {Major := maj;};
```

```
        getMajor(): MajMinType READONLY;
        { RETURN Major; };

        setYear(year : YearType) READONLY;
        {Year := year;};
        getYear(): YearType READONLY;
        { RETURN Year; };

        setSupervisor_facm(superfac : faculty_memberClass) READONLY;
        {Supervisor_facm := superfac;};
        getSupervisor_facm(): faculty_memberClass READONLY;
        { RETURN Supervisor_facm; };

        create_undergrad(facmem:faculty_memberClass,Proj:STRING,
        majr:MajMinType) READONLY;
                {

                   VAR sid2      : INT;
                   VAR sid        : INT;

                   SELF->setSupervisor_facm(facmem);
                   SELF->SYS_set_Project(Proj);
                   SELF->SYS_set_Major(majr);
                   sid2 := SELF->SYS_StudentId();
                   endline();
                   prints('student id from undergrad '); printi(sid2);
   endline();

                };
END;


CLASS   undergrad       METACLASS CATEGORY_SPECIALIZATION_CLASS

        INSTTYPE undergradType [ undergrad, faculty_member, student,
                        students, student_union, transcript, person]

        INIT    undergrad->defCategory( student, aspect )

END;

// 21. class grad
```

```
OBJECTTYPE gradType [ gradClass : gradType,
                      professorClass : professorType,
                      studentClass : studentType,
                      studentsClass : studentsType,
                      student_unionClass : student_unionType,
                      transcriptClass : transcriptType,
                      personClass : personType]
                      SUBTYPEOF studentType [ studentClass,
                                              studentsClass,
                                              student_unionClass,
                                              transcriptClass,
                                              personClass];

      INTERFACE
        PROPERTIES
                Major : MajorType;
                UnderMinor : MajMinType;
                UnderMajor : MajMinType;
                Supervisor_pr : professorClass;
        METHODS
                //setMajor(major : MajorType) READONLY;
                //getMajor(): MajorType READONLY;
                setUnderMinor(unminor : MajMinType) READONLY;
                getUnderMinor(): MajMinType READONLY;
                setUnderMajor(unmajor : MajMinType) READONLY;
                getUnderMajor(): MajMinType READONLY;
                setSupervisor_pr(superpr : professorClass) READONLY;
                getSupervisor_pr(): professorClass READONLY;
                create_grad(supervisor:professorClass) READONLY;

        IMPLEMENTATION
EXTERN prints(s:STRING);
        EXTERN printi(i:INT);
        EXTERN printr(v:REAL);
        EXTERN endline();
        METHODS

                //setMajor(major : MajorType) READONLY;
                //{Major := major;};
                //getMajor(): MajorType READONLY;
                //{ RETURN Major; };
```

```
                setUnderMinor(unminor : MajMinType) READONLY;
                {UnderMinor := unminor;};
                getUnderMinor(): MajMinType READONLY;
                { RETURN UnderMinor; };

                setUnderMajor(unmajor : MajMinType) READONLY;
                {UnderMajor := unmajor;};
                getUnderMajor(): MajMinType READONLY;
                { RETURN UnderMajor; };

                setSupervisor_pr(superpr : professorClass) READONLY;
                {Supervisor_pr := superpr;};
                getSupervisor_pr(): professorClass READONLY;
                { RETURN Supervisor_pr; };

                create_grad(supervisor:professorClass) READONLY;
                {
                   VAR sid2 :INT;
                   SELF->setSupervisor_pr(supervisor);
                   sid2 := SELF->SYS_StudentId();
                   endline();
                   prints('student id from undergrad ');
   printi(sid2); endline();
                };
END;


CLASS   grad    METACLASS CATEGORY_SPECIALIZATION_CLASS

        INSTTYPE gradType [ grad, professor, student, students,
                            student_union,transcript, person]

        INIT    grad->defCategory( student, aspect )

END;


// 26. class faculty_member

OBJECTTYPE faculty_memberType [ faculty_memberClass : faculty_memberType,
                                instructorClass : instructorType,
```

```
                                instructorsClass : instructorsType,
                                sectionsClass : sectionsType,
                                employeesClass : employeesType,
                                employeeClass : employeeType,
                                resumeClass : resumeType,
                                personClass : personType ]
                        SUBTYPEOF instructorType [ instructorClass,
                                                   instructorsClass,
                                                   sectionsClass,
                                                   employeesClass,
                                                   employeeClass,
                                                   resumeClass,
                                                   personClass];

      INTERFACE
        PROPERTIES
                InsPolNum : STRING;
                OfficeHour : STRING;

        METHODS
create_facultymem(inspolnum:STRING,officehr:STRING) READONLY;
        IMPLEMENTATION
        EXTERN prints(s:STRING);
        EXTERN printi(i:INT);
        EXTERN printr(v:REAL);
        EXTERN endline();
        METHODS
        create_facultymem(inspolnum:STRING,officehr:STRING) READONLY;
        {
                  SELF->SYS_set_InsPolNum(inspolnum);
                  SELF->SYS_set_OfficeHour(officehr);

        };
END;


CLASS   faculty_member      METACLASS CATEGORY_SPECIALIZATION_CLASS

        INSTTYPE faculty_memberType [ faculty_member,  instructor,
                        instructors, sections,
                                employees, employee, resume, person ]
```

```
        INIT    faculty_member->defCategory(instructor, aspect)
END;

// 29. class professor


OBJECTTYPE professorType [ professorClass : professorType,
                           professorsClass : professorsType,
                           studentsClass : studentsType,
                           faculty_memberClass : faculty_memberType,
                           instructorClass : instructorType,
                           instructorsClass : instructorsType,
                           sectionsClass : sectionsType,
                           employeesClass : employeesType,
                           employeeClass : employeeType,
                           resumeClass : resumeType,
                           personClass : personType]
                 SUBTYPEOF faculty_memberType [ faculty_memberClass,
                                                instructorClass,
                                                instructorsClass,
                                                sectionsClass,
                                                employeesClass,
                                                employeeClass,
                                                resumeClass,
                                                personClass ];

      INTERFACE
        PROPERTIES
                SpecialArea : { STRING };
                Supervisees1 : studentsClass;
                memberof_pros : professorsClass;
        METHODS
                setSpecialArea(spearea : { STRING }) READONLY;
                getSpecialArea(): { STRING } READONLY;
                setSupervisees1(super1 : studentsClass) READONLY;
                getSupervisees1(): studentsClass READONLY;
                setmemberof_pros(memof_pros : professorsClass) READONLY;
                getmemberof_pros(): professorsClass READONLY;
                create_prof(supervisees:studentsClass,
                memof_pros:professorsClass) READONLY;

        IMPLEMENTATION
```

```
        EXTERN prints(s:STRING);
        EXTERN printi(i:INT);
        EXTERN printr(v:REAL);
        EXTERN endline();
        METHODS
                setSpecialArea(spearea : { STRING }) READONLY;
                {SpecialArea := spearea;};
                getSpecialArea(): { STRING } READONLY;
                { RETURN SpecialArea; };

                setSupervisees1(super1 : studentsClass) READONLY;
                {Supervisees1 := super1;};
                getSupervisees1(): studentsClass READONLY;
                { RETURN Supervisees1; };

                setmemberof_pros(memof_pros : professorsClass) READONLY;
                {memberof_pros := memof_pros;};
                getmemberof_pros(): professorsClass READONLY;
                { RETURN memberof_pros; };
                create_prof(supervisees:studentsClass,
                memof_pros:professorsClass) READONLY;
                {
                   SELF->setSupervisees1(supervisees);
                   SELF->setmemberof_pros(memof_pros);
                };
END;


CLASS   professor      METACLASS CATEGORY_SPECIALIZATION_CLASS

        INSTTYPE professorType [ professor, professors, students,
                        faculty_member, instructor, instructors,
                        sections, employees, employee, resume, person ]

        INIT    professor->defCategory( faculty_member, aspect )

END;


// 30. class professorsType
```

```
OBJECTTYPE professorsType [ professorsClass : professorsType,
                            professorClass : professorType ]
                    SUBTYPEOF Metaclass_InstType;

      INTERFACE
        PROPERTIES
                Purpose : STRING;
                NumOfProfessors : INT;
                setof :  professorClass ;
        METHODS
                setsetof(stof :  professorClass ) READONLY;
                getsetof():  professorClass  READONLY;
                create_profs(purpose:STRING,numofprofs:INT,
                setprofs:professorsClass) READONLY;

        IMPLEMENTATION
        EXTERN prints(s:STRING);
        EXTERN printi(i:INT);
        EXTERN printr(v:REAL);
        EXTERN endline();
        METHODS
                setsetof(stof :  professorClass ) READONLY;
                {setof := stof;};
                getsetof():  professorClass  READONLY;
                { RETURN setof; };
                create_profs(purpose:STRING,numofprofs:INT,
                setprofs:professorsClass) READONLY;
                {
                   SELF->SYS_set_Purpose(purpose);
                   SELF->SYS_set_NumOfProfessors(numofprofs);
                };
END;

CLASS   professors      METACLASS CATEGORY_SPECIALIZATION_CLASS

        INSTTYPE professorsType [ professors, professor]

END;

// 31. class phd_advisor

OBJECTTYPE phd_advisorType [ phd_advisorClass : phd_advisorType,
```

```
                                professorClass : professorType,
                                professorsClass : professorsType,
                                studentsClass : studentsType,
                                faculty_memberClass : faculty_memberType,
                                instructorClass : instructorType,
                                instructorsClass : instructorsType,
                                sectionsClass : sectionsType,
                                employeesClass : employeesType,
                                employeeClass : employeeType,
                                resumeClass : resumeType,
                                personClass : personType]
                           SUBTYPEOF professorType[ professorClass,
                                                    professorsClass,
                                                    studentsClass,
                                                    faculty_memberClass,
                                                    instructorClass,
                                                    instructorsClass,
                                                    sectionsClass,
                                                    employeesClass,
                                                    employeeClass,
                                                    resumeClass,
                                                    personClass ];

      INTERFACE
        PROPERTIES
              Title : STRING;
        METHODS
create_phdadvisor(title:STRING) READONLY;
        IMPLEMENTATION
        EXTERN prints(s:STRING);
        EXTERN printi(i:INT);
        EXTERN printr(v:REAL);
        EXTERN endline();
        METHODS
                    create_phdadvisor(title:STRING) READONLY;
                    {
                    SELF->SYS_set_Title(title);
                    };
END;
CLASS   phd_advisor     METACLASS CATEGORY_SPECIALIZATION_CLASS

        INSTTYPE phd_advisorType [ phd_advisor, professor, professors,
```

```
                          students,  faculty_member, instructor,
   instructors, sections,
                          employees, employee, resume, person]


        INIT    phd_advisor->defCategory( professor, aspect )

END;

// 39. class instructors

OBJECTTYPE instructorsType [ instructorsClass : instructorsType,
                             instructorClass : instructorType ]
                              SUBTYPEOF Metaclass_InstType;

      INTERFACE
        PROPERTIES
                Purpose : STRING;
                NumOfFacultyMembers : INT;
                setof: instructorClass;
        METHODS
                setsetof(stof: instructorClass) READONLY;
                getsetof(): instructorClass READONLY;


        IMPLEMENTATION
        EXTERN prints(s:STRING);
        EXTERN printi(i:INT);
        EXTERN printr(v:REAL);
        EXTERN endline();
        METHODS
                setsetof(stof: instructorClass) READONLY;
                {setof := stof;};
                getsetof(): instructorClass READONLY;
                { RETURN setof; };
END;


CLASS   instructors

        INSTTYPE instructorsType [ instructors, instructor ]
```

```
END;

END_SCHEMA;
```

APPENDIX B

APPLICATION PROGRAMS

B.1 Create Database Program

```
//=========================================================================
//              Standard header files
//=========================================================================
#include <stdio.h>


//=========================================================================
//              Parallax header files
//=========================================================================
//#include <Exception.h>
//#include <nihclIO.h>
//#include <nihclerrs.h>

//=========================================================================
//              VODAK header files
//=========================================================================
//#include "/home/earth/grad/klas/newvodak/VODAK/RTE/vml.h"
//#include "/home/earth/grad/klas/newvodak/VODAK/UT/Debug.h"
//#include "/home/earth/grad/klas/newvodak/VODAK/DD/SysCat.h"
//#include "/home/earth/grad/klas/newvodak/VODAK/DD/METHODS.h"

#include <UT/Debug.h>
#include <DD/SysCat.h>
#include <DD/METHODS.h>
#include <RTE/vml.h>

extern "C" {

//=========================================================================
//              Extern variables declaration
//=========================================================================
//TID *tid;
```

```
//=========================================================================
//              VODAK application program
//=========================================================================
void program() {

   TID *tid = new TID();

   //=================================================
   //        create a local database
   //=================================================

//   CREATEDATABASE("db1", 0, "twentyschema", "1.00");

//   C++ code fore CREATEDATABASE

     VmlOid *systemSchemaOid = (VmlOid*) SEND (tid, TheVmlSchemaOid, 3,
                           MethSel("detect"),
                           (VmlDataType*) (new VmlString("name")),
                           (VmlDataType*) (new VmlString("==")),
                           (VmlDataType*) (new VmlString ("SYSTEM")));

     SEND (tid, systemSchemaOid, 1, MethSel("createDatabase"),
           (VmlDataType*) (new VmlString ("SYSTEM")));

     VmlOid *twentyschema = (VmlOid*) SEND (tid, TheVmlSchemaOid, 3,
                            MethSel("detect"),
                            (VmlDataType*) (new VmlString("name")),
                            (VmlDataType*) (new VmlString("==")),
                            (VmlDataType*) (new VmlString ("twentyschema")));

     SEND (tid, twentyschema, 1, MethSel("createDatabase"),
                  (VmlDataType*) (new VmlString ("db1")));

}

} //extern C
```

B.2 Application Program

```
//=============================================================================
//              Standard header files
//=============================================================================
#include <iostream.h>

//=============================================================================
//              Parallax header files
//=============================================================================
//#include <Exception.h>
//#include <nihclIO.h>
//#include <nihclerrs.h>

//=============================================================================
//              VODAK header files
//=============================================================================
//#include "/home/earth/grad/klas/newvodak/VODAK/RTE/vml.h"
//#include "/home/earth/grad/klas/newvodak/VODAK/UT/Debug.h"
//#include "/home/earth/grad/klas/newvodak/VODAK/DD/SysCat.h"
//#include "/home/earth/grad/klas/newvodak/VODAK/DD/METHODS.h"

//#include <UT/Debug.h>
//#include <DD/SysCat.h>
//#include <DD/METHODS.h>
#include <RTE/vml.h>

extern "C" {

//=============================================================================
//              Extern variables declaration
//=============================================================================
//TID *tid;


//=============================================================================
//              VODAK application program
//=============================================================================
void program() {

   VmlOid *p_person1;
```

```
VmlOid *p_person2;
VmlOid *p_person3;
VmlOid *p_person4;
VmlOid *p_person5;
VmlOid *p_person6;

VmlOid *s_student1;
VmlOid *s_student2;
VmlOid *s_student3;
VmlOid *s_student4;
VmlOid *s_student5;
VmlOid *stud_1, *stud_2;
VmlOid *stud_set;
VmlSet stud_s1;
VmlSet stud_s2;

VmlOid *e_emp1;
VmlOid *e_emps2;
VmlOid *e_emp2;

VmlOid *i_ins1;
VmlOid *i_ins2;

VmlOid *s_sec1;
VmlOid *s_secs;

VmlOid *crsec;

VmlOid *c_crse;

VmlOid *s_union1;

VmlOid *t_trans, *t_trans1;

VmlOid *r_res;

VmlOid *u_grad;

VmlOid *facm;

VmlOid *prof;
```

```
   VmlOid *memofprofs;

   VmlOid *s_visees;

   VmlSet instance_set;

   VmlOid *g_grad;

   VmlString test;
   VmlInteger i, k;
   VmlReal d;
   VmlString n, m, l;

   VmlString return_string;
   Iterator it(stud_s1);
   Iterator count(instance_set);

   TID *tid = new TID();
   char *classname;
   int loop, choice;

   //=====================================================
   //        open local database
   //=====================================================

//   OPENDATABASE("db1", "twentyschemA", "1.00");

//   C++ code for OPENDATABASE

     VmlOid *systemDBOid = (VmlOid*) SEND (tid, TheVmlDatabaseOid, 3,
                          MethSel("detect"),
                          (VmlDataType*) (new VmlString("name")),
                          (VmlDataType*) (new VmlString("==")),
                          (VmlDataType*) (new VmlString ("SYSTEM")));

     SEND (tid, systemDBOid, 0, MethSel("openDatabase"));

     VmlOid *db1Oid = (VmlOid*) SEND (tid, TheVmlDatabaseOid, 3,
                            MethSel("detect"),
                            (VmlDataType*) (new VmlString("name")),
                            (VmlDataType*) (new VmlString("==")),
                            (VmlDataType*) (new VmlString ("db1")));
```

```
    SEND (tid, db1Oid, 0, MethSel("openDatabase"));

  //=================================================
  //     create an instance
  //=================================================

  loop = 1;
  while(loop) {

      cout << "Please choose one of the options:" << endl;
      cout << "   1 ..... Creating  the  default instances" << endl;
      cout << "   3 ..... Print all persons " << endl;
      cout << " " << endl;
      cout << "   0 ..... Quit" << endl;
      cout << " " << endl;

      cout << "Your choice : ";
      cin >> choice;

      switch (choice)
      {

  //=====================================================
  //     creating  default instances
  //=====================================================
         case 1:  cout << "Creating a person ....." << endl;
                  p_person1 = (VmlOid *)SEND(tid, CLSID("person"), 0,
                                  MethSel("new"));
                  SEND (tid, p_person1, 4, MethSel("create_person"),
                       (VmlDataType*) (new VmlString ("g1")),
                       (VmlDataType*) (new VmlString ("harison")),
                       (VmlDataType*) new VmlInteger(78785),
                       (VmlDataType*) (new VmlString ("madhu")));

                  cout << "created 1stperson " << endl;
//=================================================
//     create an instance another person
//=================================================
                  p_person2 = (VmlOid *)SEND(tid, CLSID("person"), 0,
                                  MethSel("new"));
                  SEND (tid, p_person2, 4, MethSel("create_person"),
```

```
                    (VmlDataType*) (new VmlString ("h1")),
                    (VmlDataType*) (new VmlString ("kearny")),
                    (VmlDataType*) new VmlInteger(78986),
                    (VmlDataType*) (new VmlString ("aruna")));

               cout << "created 2nd person " << endl;
//=================================================
//     create an instance another person
//=================================================
               p_person3 = (VmlOid *)SEND(tid, CLSID("person"), 0,
                                   MethSel("new"));
               SEND (tid, p_person3, 4, MethSel("create_person"),
                    (VmlDataType*) (new VmlString ("h1")),
                    (VmlDataType*) (new VmlString ("kearny")),
                    (VmlDataType*) new VmlInteger(78986),
                    (VmlDataType*) (new VmlString ("aruna")));

               cout << "created 3rd person " << endl;
//=================================================
//     create an instance another person
//=================================================
               p_person4 = (VmlOid *)SEND(tid, CLSID("person"), 0,
                                   MethSel("new"));
               SEND (tid, p_person4, 4, MethSel("create_person"),
                    (VmlDataType*) (new VmlString ("h1")),
                    (VmlDataType*) (new VmlString ("kearny")),
                    (VmlDataType*) new VmlInteger(78986),
                    (VmlDataType*) (new VmlString ("aruna")));

               cout << "created 4th person " << endl;
//=================================================
//     create an instance another person
//=================================================
               p_person5 = (VmlOid *)SEND(tid, CLSID("person"), 0,
                                   MethSel("new"));
               SEND (tid, p_person5, 4, MethSel("create_person"),
                    (VmlDataType*) (new VmlString ("h1")),
                    (VmlDataType*) (new VmlString ("kearny")),
                    (VmlDataType*) new VmlInteger(78986),
                    (VmlDataType*) (new VmlString ("aruna")));

               cout << "created 5th person " << endl;
```

```
//=================================================
//     create an instance another person
//=================================================
                    p_person6 = (VmlOid *)SEND(tid, CLSID("person"), 0,
                                        MethSel("new"));
                    SEND (tid, p_person6, 4, MethSel("create_person"),
                         (VmlDataType*) (new VmlString ("h1")),
                         (VmlDataType*) (new VmlString ("kearny")),
                         (VmlDataType*) new VmlInteger(78986),
                         (VmlDataType*) (new VmlString ("aruna")));

                    cout << "created 6th person " << endl;
//=================================================
//     create an instance of a first student
//=================================================
                    cout << "Creating a person ....." << endl;
                    p_person1 = (VmlOid *)SEND(tid, CLSID("person"), 0,
                                        MethSel("new"));

                    cout << "Creating a student ....." << endl;
                    s_student1 = (VmlOid *)SEND(tid, CLSID("student"), 0,
                                        MethSel("new"));
//=================================================
//     create an instance of a transcript
//=================================================
                cout << "Creating a transcript ....." << endl;
                    t_trans = new VmlOid (*
        (VmlOid *)SEND(tid, CLSID("transcript"), 0,
                                        MethSel("new")));

//                  set category relation

                    SEND (tid, s_student1, 1, MethSel("initCategoryOf"),
                                    (VmlOid *) p_person1);
                    SEND (tid, s_student1, 6, MethSel("Create_student"),
                         (VmlDataType*) (new VmlString ("MIS")),
                         (VmlDataType*) (new VmlString ("harison")),
                         (VmlDataType*) (new VmlString ("bachlers")),
                         (VmlDataType*) new VmlInteger(5),
              (VmlOid *) t_trans,
                         (VmlDataType*) (new VmlString ("cross")));
```

```
                    cout << "created 1st student " << endl;
//=================================================
//      create an instance 2nd student
//=================================================
                    p_person2 = (VmlOid *)SEND(tid, CLSID("person"), 0,
                                        MethSel("new"));
                    cout << "Creating a another student ....." << endl;
                    s_student2 = (VmlOid *)SEND(tid, CLSID("student"), 0,
                                        MethSel("new"));
//=================================================
//      create an instance of a transcript
//=================================================
                cout << "Creating a transcript ....." << endl;
                    t_trans1 = new VmlOid (*
        (VmlOid *)SEND(tid, CLSID("transcript"), 0,
                                        MethSel("new")));

//                  set category relation

                    SEND (tid, s_student2, 1, MethSel("initCategoryOf"),
                                    (VmlOid *) p_person2);
                    SEND (tid, s_student2, 6, MethSel("Create_student"),
                         (VmlDataType*) (new VmlString ("CIS")),
                         (VmlDataType*) (new VmlString ("kearny")),
                         (VmlDataType*) (new VmlString ("Graduate")),
                         (VmlDataType*) new VmlInteger(787),
             (VmlOid *) t_trans,
                         (VmlDataType*) (new VmlString ("Newjersey")));

                    cout << "created 2nd student " << endl;
//=================================================
//      create an instance 3rd student
//=================================================
                    p_person3 = (VmlOid *)SEND(tid, CLSID("person"), 0,
                                        MethSel("new"));
                    cout << "Creating a another student ....." << endl;
                    s_student3 = (VmlOid *)SEND(tid, CLSID("student"), 0,
                                        MethSel("new"));
//=================================================
//      create an instance of a transcript
//=================================================
                cout << "Creating a transcript ....." << endl;
```

```
                    t_trans1 = new VmlOid (*
        (VmlOid *)SEND(tid, CLSID("transcript"), 0,
                                    MethSel("new")));

//                  set category relation

                    SEND (tid, s_student3, 1, MethSel("initCategoryOf"),
                                    (VmlOid *) p_person3);
                    SEND (tid, s_student3, 6, MethSel("Create_student"),
                         (VmlDataType*) (new VmlString ("MIS")),
                         (VmlDataType*) (new VmlString ("georgia")),
                         (VmlDataType*) (new VmlString ("PhD")),
                         (VmlDataType*) new VmlInteger(7987),
              (VmlOid *) t_trans,
                         (VmlDataType*) (new VmlString ("Athens")));

                    cout << "created 3rd student " << endl;
   //=================================================
   //     create an instance 4th student
   //=================================================
                    p_person4 = (VmlOid *)SEND(tid, CLSID("person"), 0,
                                    MethSel("new"));
               cout << "Creating a another student ....." << endl;
                    s_student4 = (VmlOid *)SEND(tid, CLSID("student"), 0,
                                    MethSel("new"));
   //=================================================
   //     create an instance of a transcript
   //=================================================
               cout << "Creating a transcript ....." << endl;
                    t_trans1 = new VmlOid (*
        (VmlOid *)SEND(tid, CLSID("transcript"), 0,
                                    MethSel("new")));

//                  set category relation

                    SEND (tid, s_student4, 1, MethSel("initCategoryOf"),
                                    (VmlOid *) p_person4);
                    SEND (tid, s_student4, 6, MethSel("Create_student"),
                         (VmlDataType*) (new VmlString ("Astronomy")),
                         (VmlDataType*) (new VmlString ("India")),
                         (VmlDataType*) (new VmlString ("Masters")),
                         (VmlDataType*) new VmlInteger(9654),
```

```
            (VmlOid *) t_trans,
                        (VmlDataType*) (new VmlString ("Secunderabad")));

                    cout << "created 4th student " << endl;
    //=================================================
    //      create an instance 5th student
    //=================================================
                    p_person5 = (VmlOid *)SEND(tid, CLSID("person"), 0,
                                        MethSel("new"));
                    cout << "Creating a another student ....." << endl;
                    s_student5 = (VmlOid *)SEND(tid, CLSID("student"), 0,
                                        MethSel("new"));
    //=================================================
    //      create an instance of a transcript
    //=================================================
                cout << "Creating a transcript ....." << endl;
                    t_trans1 = new VmlOid (*
        (VmlOid *)SEND(tid, CLSID("transcript"), 0,
                                        MethSel("new")));

//                  set category relation

                    SEND (tid, s_student5, 1, MethSel("initCategoryOf"),
                                    (VmlOid *) p_person5);
                    SEND (tid, s_student5, 6, MethSel("Create_student"),
                         (VmlDataType*) (new VmlString ("CIS")),
                         (VmlDataType*) (new VmlString ("kearny")),
                         (VmlDataType*) (new VmlString ("Graduate")),
                         (VmlDataType*) new VmlInteger(787),
               (VmlOid *) t_trans,
                         (VmlDataType*) (new VmlString ("Newjersey")));

                    cout << "created 5th student " << endl;
    //=================================================
    //      create a set of students
    //=================================================
        //stud_1 = (VmlOid *)SEND(tid, CLSID("student"), 0,
                                        //MethSel("new"));
                    //stud_2 = (VmlOid *)SEND(tid, CLSID("student"), 0,
                                        //MethSel("new"));
                    stud_set = new VmlOid (*
        (VmlOid *)SEND(tid, CLSID("students"), 0,
```

```
                                        MethSel("new")));
                    stud_s1.add(*s_student1); // Insert stud_1 into stud_s1
                    stud_s1.add(*s_student2);
                    stud_s1.add(*s_student3);
                    stud_s1.add(*s_student4);
                    stud_s1.add(*s_student5);
                    SEND(tid, stud_set, 1, MethSel("setsetof"),
                                (VmlSet) stud_s1);

                    //stud_s2 = *(VmlSet *)SEND(tid, stud_set, 0,
                                        //MethSel("getsetof"));
                    cout << "created an instance of students " << endl;

   //=================================================
   //     printing the set of students using iterator
   //=================================================
   while(it++)
                    {
                      VmlOid t1 = *VmlOid::castdown(it());
                      test = *(VmlString *)SEND(tid, &t1, 0,
                                        MethSel("SYS_Degree"));
                      cout << "Name of student education -- " << test << endl;
                    }
//=================================================
//     create an instance of a employee
//=================================================
          cout << "Creating a person ....." << endl;
                    p_person6 = (VmlOid *)SEND(tid, CLSID("person"), 0,
                                        MethSel("new"));

               cout << "Creating a employee ....." << endl;
                    e_emp1 = (VmlOid *)SEND(tid, CLSID("employee"), 0,
                                        MethSel("new"));
//=================================================
//     create an instance of a resume
//=================================================
                    cout << "Creating a resume ....." << endl;
                    r_res = new VmlOid (*
        (VmlOid *)SEND(tid, CLSID("resume"), 0,
                                        MethSel("new")));

//                  set category relation
```

```
                    SEND (tid, e_emp1, 1, MethSel("initCategoryOf"),
                                    (VmlOid *) p_person6);
                    SEND (tid, e_emp1, 7, MethSel("create_emp"),
                         (VmlDataType*) new VmlInteger(1),
                         (VmlDataType*) new VmlInteger(40),
                         (VmlDataType*) (new VmlString ("Manhatten")),
                         (VmlDataType*) new VmlInteger(5),
               (VmlDataType*) new VmlReal(2.5),
                         (VmlDataType*) (new VmlString ("8989")),
               (VmlOid *) r_res);

                    cout << "created employee " << endl;
//=================================================
//     create an instance of a StudentUnion
//=================================================
          cout << "Creating a StudentUnion ....." << endl;
                    e_emps2 = new VmlOid (*
        (VmlOid *)SEND(tid, CLSID("employees"), 0,
                                    MethSel("new")));
                    s_union1 = (VmlOid *)SEND(tid, CLSID("student_union"), 0,
                                    MethSel("new"));
                    SEND (tid, s_union1, 3, MethSel("create_studunion"),
                         (VmlDataType*) (new VmlString ("Wallstreet")),
                         (VmlDataType*) new VmlInteger(785),
               (VmlOid *) e_emps2);

                    cout << "created StudentUnion " << endl;
//=================================================
//     create an instance of a Section
//=================================================
          cout << "Creating a Section ....." << endl;
                    i_ins1 = new VmlOid (*
        (VmlOid *)SEND(tid, CLSID("instructor"), 0,
                                    MethSel("new")));
                    s_sec1 = (VmlOid *)SEND(tid, CLSID("section"), 0,
                                    MethSel("new"));
                    SEND (tid, s_sec1, 3, MethSel("create_secs"),
                         (VmlDataType*) new VmlInteger(8),
                         (VmlDataType*) new VmlInteger(8),
               (VmlOid *) i_ins1);
```

```
                    cout << "created section " << endl;
//=================================================
//     create an instance of a CrSection
//=================================================
          cout << "Creating a CrSection ....." << endl;
                    c_crse = new VmlOid (*
        (VmlOid *)SEND(tid, CLSID("course"), 0,
                                    MethSel("new")));
                    crsec = (VmlOid *)SEND(tid, CLSID("crsections"), 0,
                                    MethSel("new"));
                    SEND (tid, crsec, 2, MethSel("create_crsec"),
                         (VmlDataType*) new VmlInteger(8),
             (VmlOid *) c_crse);

                    cout << "created section " << endl;
//=================================================
//     create an instance of a course
//=================================================
          cout << "Creating a course ....." << endl;
                    crsec = new VmlOid (*
        (VmlOid *)SEND(tid, CLSID("crsections"), 0,
                                    MethSel("new")));
                    c_crse = (VmlOid *)SEND(tid, CLSID("course"), 0,
                                    MethSel("new"));
                    SEND (tid, c_crse, 5, MethSel("create_course"),
                         (VmlDataType*) (new VmlString ("785")),
                         (VmlDataType*) (new VmlString ("CIS")),
                         (VmlDataType*) new VmlInteger(3),
                         (VmlDataType*) new VmlInteger(8),
             (VmlOid *) crsec);

                    cout << "created course " << endl;
//=================================================
//     create an instance of a instructor
//=================================================
          cout << "Creating a instructor ....." << endl;
                    e_emp2 = (VmlOid *)SEND(tid, CLSID("employee"), 0,
                                    MethSel("new"));
                    s_secs = new VmlOid (*
        (VmlOid *)SEND(tid, CLSID("sections"), 0,
                                    MethSel("new")));
                    i_ins2 = (VmlOid *)SEND(tid, CLSID("instructor"), 0,
```

```
                                        MethSel("new"));
//                  set category relation

                    SEND (tid, i_ins2, 1, MethSel("initCategoryOf"),
                                    (VmlOid *) e_emp2);
                    SEND (tid, i_ins2, 2, MethSel("Create_instructor"),
            (VmlOid *) s_secs,
            (VmlDataType*) new VmlReal(2.5));

                    cout << "created instructor " << endl;
//=================================================
//     create an instance of a undergrad
//=================================================
          cout << "Creating a undergrad ....." << endl;
                    s_student1 = (VmlOid *)SEND(tid, CLSID("student"), 0,
                                        MethSel("new"));
                    facm = new VmlOid (*
       (VmlOid *)SEND(tid, CLSID("faculty_member"), 0,
                                        MethSel("new")));
                    u_grad = (VmlOid *)SEND(tid, CLSID("undergrad"), 0,
                                        MethSel("new"));
//                  set category relation

                    SEND (tid, u_grad, 1, MethSel("initCategoryOf"),
                                    (VmlOid *) s_student1);
                    SEND (tid, u_grad, 3, MethSel("create_undergrad"),
            (VmlOid *) facm,
                         (VmlDataType*) (new VmlString ("OODB")),
                         (VmlDataType*) (new VmlString ("CIS")));

                    cout << "created undergrad " << endl;
//=================================================
//     create an instance of a grad
//=================================================
          cout << "Creating a grad ....." << endl;
                    s_student2 = (VmlOid *)SEND(tid, CLSID("student"), 0,
                                        MethSel("new"));
                    prof = new VmlOid (*
       (VmlOid *)SEND(tid, CLSID("professor"), 0,
                                        MethSel("new")));
                    g_grad = (VmlOid *)SEND(tid, CLSID("grad"), 0,
                                        MethSel("new"));
```

```
//                  set category relation

                    SEND (tid, g_grad, 1, MethSel("initCategoryOf"),
                                    (VmlOid *) s_student2);
                    SEND (tid, g_grad, 1, MethSel("create_grad"),
             (VmlOid *) prof);

                    cout << "created grad " << endl;
//=================================================
//     create an instance of a professor
//=================================================
           cout << "Creating a professor ....." << endl;
                    facm = (VmlOid *)SEND(tid, CLSID("faculty_member"), 0,
                                    MethSel("new"));
                    s_visees = new VmlOid (*
            (VmlOid *)SEND(tid, CLSID("students"), 0,
                                    MethSel("new")));
                    memofprofs = new VmlOid (*
        (VmlOid *)SEND(tid, CLSID("professors"), 0,
                                    MethSel("new")));
                    prof = (VmlOid *)SEND(tid, CLSID("professor"), 0,
                                    MethSel("new"));
//                  set category relation

                    SEND (tid, prof, 1, MethSel("initCategoryOf"),
                                    (VmlOid *) facm);
                    SEND (tid, prof, 2, MethSel("create_prof"),
              (VmlOid *) s_visees,
              (VmlOid *) memofprofs);

                    cout << "created professor " << endl;
//=================================================
// Printing all persons
//=================================================
   case 2:  cout << "Printing all persons" << endl;

                    instance_set = *(VmlSet *)SEND(tid, CLSID("person"), 0,
                                    MethSel("allInstances"));

                    while(count++)
                    {
                       VmlOid t1 = *VmlOid::castdown(count());
```

```
                    test = *(VmlString *)SEND(tid, &t1, 0,
                                              MethSel("getPerData"));
                    cout << "name of person -- " << test << endl;
                }
                break;
//=================================================
        case 0: loop = 0;
break;
    }
}
}

} //extern C
```

APPENDIX C

C++ CLASS LIBRARY SOURCE LISTING OF GUI

```
//////////////////////////////////////////////////////////////////////////
// Class : MainWin
//////////////////////////////////////////////////////////////////////////
// Purpose    : To create the container widget to primarily to handle the
// work area and menubar
//
// Author     : Bheeman
//
// Date       : Nov December 6, 1992
//
// Synopsis   : Create a popup shell widget to contain Motif XmMainWindow
//              widget and create XmMainWindow widget to handle layout of
//              a work area and a menubar. The application needs to create
//              only the widgets that appear in the work area.
//
//////////////////////////////////////////////////////////////////////////
#ifndef MAINWIN_H
#define MAINWIN
#include "Iface.h"
class MainWin : public Iface {
 protected:

         Widget _main;
         Widget _workArea;
         virtual Widget createWorkArea ( Widget ) = 0;
 public:

         MainWin (char *);
         virtual ~MainWin();
          virtual void initialize();
          virtual void manage();
          virtual void unmanage();
          virtual void iconify();
};
#endif
```

```
////////////////////////////////////////////////////
//   Implementation of Class MainWin              //
////////////////////////////////////////////////////

#include "Frame.h"
#include "MainWin.h"
#include <Xm/MainW.h>
#include <assert.h>
#include <X11/StringDefs.h>
MainWin::MainWin ( char *name ) : Iface ( name )
{
  _workArea = NULL;
   assert ( theFrame );
         // The application must create Frame object before
         // the MainWindow is  created
   theFrame->registerWindow ( this );
}

void MainWin::initialize()
{
  _w = XtCreatePopupShell ( _name,
                        applicationShellWidgetClass,
                        theFrame->rootWidget(),
                                 NULL, 0 );

  installDestroyHandler();

  _main = XtCreateManagedWidget ( "mainwindow",
  xmMainWindowWidgetClass,
   _w,
   NULL,
   0 );

  _workArea = createWorkArea ( _main );
  // call derived class's function to create
   // application specific work area
   assert ( _workArea );

  XtVaSetValues (_main,
                 XmNworkWindow, _workArea,
                 XmNwidth,
```

```
                    900,
                    XmNheight,
                    800,
                    NULL
  );
  // make the work area widget XmNworkArea
  // widget of XmMainWindow widget
 if ( !XtIsManaged ( _workArea )
          XtManageChild (_workArea );
  // Manage the work area of the application
 // if not managed already
}

MainWin::~MainWin()
{
 theFrame->unregisterWindow ( this);
  // removes the deleted object from Frame
  // object's list of windows

 }
// manage() function overrides Iface class's manage()
//function because the popup shell created by MainWin class
//must be made visible with XtPopup()
// instead of XtManageChild()
 void MainWin::manage()
 {
  assert ( _w );
   XtPopup ( _w, XtGrabNone );
   if ( XtIsRealized ( _w ) )
    XMapRaised ( XtDisplay ( _w ), XtWindow ( _w ) );
}

// unmanage() function overrides Iface class's unmanage() function because
// XtPOpdown() function needs to be called to remove popup from screen,
// instead of XtUnmanageChild()
void MainWin::unmanage()
{
  assert( _w );
  XtPopup ( _w );
  }

void MainWin::iconify()
```

```
  {
   assert ( _w );
   XtVaSetValues ( _w, XmNiconic, TRUE, NULL );
   if ( XtIsRealized ( _w ) )
          XIconifyWindow ( XtDisplay ( _w ), XtWindow ( _w ), 0 );
 }

////////////////////////////////////////////////////////////////////////
// Class  :  MenuBar
////////////////////////////////////////////////////////////////////////
// Purpose  : To create a pulldown menu from a list of Command objects
//
// Author   : Bheeman
//
// Date     : December 22, 1992
//
// Synopsis : The MenuBar class is derived from Iface class
//            It support a constructor and a member function
//            addCommands()
//
//            addCommnads()  creates a menu pane with an entry for
//            each  Command object in the list.
//
//            The constructor creates a menubar widget by calling
//            motif XmCreateMenuBar(). This widget will serve as the
//            base widget of this component
////////////////////////////////////////////////////////////////////////
#undef MENUBAR_H
#ifndef MENUBAR_H
#define MENUBAR_H
#include "Iface.h"
class Command;
class Commands;

class MenuBar : public Iface {

 public:

           MenuBar ( Widget, char * );
// Create a menu bar with the givan name
           virtual void addCommands ( Commands *, char * );
     virtual const char *const className() { return ( "MenuBar" ); }
```

```
 };
#endif


æ//////////////////////////////////////////////////
//   Implementation of Class MenuBar               //
//////////////////////////////////////////////////
#include "MenuBar.h"
#include "Command.h"
#include "Commands.h"
#include "ButtonIface.h"
#include <Xm/RowColumn.h>
#include <Xm/CascadeB.h>
MenuBar::MenuBar ( Widget parent, char *name ) : Iface ( name )
{
 _w = XmCreateMenuBar ( parent, _name, NULL, 0 );
 // create Motif menuBar widget
 installDestroyHandler();

}


void  MenuBar::addCommands ( Commands *list, char *name )
{
 int i;
 Widget pulldown, cascade;

 pulldown = XmCreatePulldownMenu ( _w, name, NULL, 0);
 // Create pulldown menu

 cascade = XtVaCreateWidget ( name,
                              xmCascadeButtonWidgetClass,
                              _w,
                               XmNsubMenuId, pulldown,
                               NULL );
   // Create cascade button

 XtManageChild ( cascade );

 for (i = 0; i < list->size(); i++)
 // Create a menu entry for each command
 {
```

```
    CmdIface *ci;
    ci = new ButtonIface ( pulldown, ( *list)[i] );
    ci->manage();
  }
}


//////////////////////////////////////////////////////////////
// Class   : MenuWin
//////////////////////////////////////////////////////////////
// Purpose : To add a menubar along the top of MainWindow
//
// Author  : Bheeman
//
// Date    : December 22, 1992
//
// Synopsis: MenuWin class is an abstract class derived from
//           MainWin class.
//
//           Applications that need to create a menubar must
//           derive a class from MenuWin.
//
//           Classes derived from MenuWin must implement the
//           pure virtual functions createMenuPanes() of this
//           class and createWorkArea() of the MainWin class
//
//           The constrauctor calls the constructor of MainWin
//           and also initializes the data member _menuBar to NULL
//
//           The initialize() member function, which overrides the
//           MainWin's initialize() first calls the MainWin's
//           initialize() to create XmMainWindow widget and to
//           setup the work area. It then creates and sets up
//           a menubar widget by invoking the MenuBar class.
//           Finally, it calls the member function, createMenuPanes()
//           of the derived class and invokes manubar object's manage()
//           function to manage the widget tree.
//
//           The destructor deletes the menubar object.
//////////////////////////////////////////////////////////////////////
#ifndef MENUWIN_H
#define MENUWIN_H
```

```
#include "MainWin.h"
class MenuBar;
class MenuWin : public MainWin {

 protected:

       MenuBar *_menuBar;
       virtual void initialize();
       virtual void createMenuPanes() = 0;

 public:

       MenuWin ( char *name );
       virtual ~MenuWin ();
};
#endif


//////////////////////////////////////////////////
//   Implementation of Class MenuWin            //
//////////////////////////////////////////////////
#include "MenuWin.h"
#include "MenuBar.h"
MenuWin::MenuWin ( char *name ) : MainWin ( name )
{

 _menuBar = NULL;

}

void MenuWin::initialize()
{

  MainWin::initialize();
  // call MainWin to create XmMainWindow and set up work area
  _menuBar = new MenuBar ( _main, "menubar");
  XtVaSetValues ( _main,
                  XmNmenuBar, _menuBar->rootWidget(),
                  NULL);

createMenuPanes();
_menuBar->manage();
```

```
}

MenuWin::~MenuWin()
{

 delete _menuBar;

}
////////////////////////////////////////////////////////////////////////
// Class  :  QuestnDialog
////////////////////////////////////////////////////////////////////////
// Purpose : To ask confirmation from the user to perform
//           a critical operation
//
// Author  : Bheeman
//
// Date    : December 24, 1992
//
// Synopsis: This class is a derived class of the DialodMgr class.
//
//           There will be only one instance of this class which
//           will be a global instance. Applications will send
//           message to this object whenever a question needs to
//           be answered.
//
//           The implementation file creates a global object of this
//           class
//
//           The constructor calls the DialogMgr's constructor.
//
//           The crealtDialog() function overrides the pure virtual
//           function of DialogMgr class. It creates a Motif Question
//           Dialog widget with dialog style set to
//           XmDIALOG_FULL_APPLICATION_MODAL
////////////////////////////////////////////////////////////////////////
#ifndef QUESTNDIALOG_H
#define QUESTNDIALOG_H
#include "DialogMgr.h"
class QuestnDialog : public DialogMgr {

 protected:
```

```
        Widget createDialog (Widget );

 public:
        QuestnDialog ( char * );

};
extern QuestnDialog *theQuestnDialog;
#endif

////////////////////////////////////////////////////
//   Implementation of Class QuestnDialog         //
////////////////////////////////////////////////////
#include "QuestnDialog.h"
#include <Xm/Xm.h>
#include <Xm/MessageB.h>
QuestnDialog *theQuestnDialog = new QuestnDialog ( "QuestnDialog" );
QuestnDialog::QuestnDialog ( char *name ) :
 DialogMgr ( name )
{

}


Widget QuestnDialog::createDialog ( Widget parent )
{
  Widget dialog = XmCreateQuestionDialog ( parent, _name, NULL, 0);
                      XtVaSetValues ( dialog,
                      XmNdialogStyle, XmDIALOG_FULL_APPLICATION_MODAL,
                      NULL );
  return ( dialog );
 }


////////////////////////////////////////////////////////////////
// class : Quit
////////////////////////////////////////////////////////////////
// Pupose  : To quit an application in userfriendly way.
//
// Author  : Bheeman
//
// Date    : January 10, 1992
//
```

```
// Synopsis: This class is derived from WarnNoUndo class. It
//           a conastructor and a doit member function.
//
//          The constructor initializes the object by calling
//          the base class's constructor and then sets the
//          default question
//
//          The doit() member funtion calls exit() system call
//          to exit from the application.
////////////////////////////////////////////////////////////
#ifndef QUITWIN_H
#define QUITWIN_H
#include "WarnNoUndo.h"
class QuitWin : public WarnNoUndo {

 protected:
          virtual void doit();

 public:
          QuitWin(char *, int);
       virtual const char *const className () { return ( "Quit" ); }

};
#endif
////////////////////////////////////////////////////
//   Implementation of Class QuitWin              //
////////////////////////////////////////////////////
#include "QuitWin.h"
#include <stdlib.h>
#define QUITQUESTION "Do you really want to quit?"
QuitWin::QuitWin ( char *name, int active ) :
               WarnNoUndo ( name, active )
{
 setQuestion ( QUITQUESTION );
}

void QuitWin::doit()
{

 exit(0);

}
```

```
////////////////////////////////////////////////////////////
//
// Class: Root.h
//
// Purpose: Defining the root class  for all application Classes //
//
// Author : Bheeman
//
// Date   : Nov 28, 1992.
//
//Synopsis: The 'Root' class is intended to be an abstract class
//which can be instanstiated by the derived classes. It defines the
//common protocol to be followed by all user components.
//
// The constructor accepts string which is the name of the base class
// class of the widget tree defined by the user component.
// _w is the base widget, which is initialized to NULL by the
//constructor
//
// The destructor destroys the base widget so that the entire
//subtree of widgets is destroyed.
//
// The manage() and unmanage() functions manage and unmanage
//the base widget of a component.
//
////////////////////////////////////////////////////////////////
#ifndef ROOT_H
#define ROOT_H
#include <Xm/Xm.h>
#include <iostream.h>
class Root {
friend main();

 protected:
         char *_name;
         Widget _w;
         Root( const char * );

 public:
         virtual ~Root();
         virtual void manage();
         virtual void unmanage();
```

```
        const Widget rootWidget() { return (_w); }
};

#endif

////////////////////////////////////////////////////
//   Implementation of Class Root                 //
////////////////////////////////////////////////////
#include "Root.h"
#include <assert.h>
#include "/usr/local/lib/g++-include/iostream.h"
//////////////////////////
//// root constructor    //
//////////////////////////
Root::Root ( const char *name) {
  _w = NULL;
  assert ( name != NULL );
  _name = strdup ( name );
}

//////////////////////////
//// root destructor ///
//////////////////////////
Root::~Root() {

  if(_w)
         XtDestroyWidget ( _w );
  delete _name;
}

////////////////////////////////
//// manage root widget        //
////////////////////////////////
void Root::manage() {
    assert ( _w != NULL );
    XtManageChild ( _w );
}

//////////////////////////////
//  unmanage widget          //
//////////////////////////////
void Root::unmanage() {
```

```
  assert ( _w != NULL );
  XtUnmanageChild ( _w );
}

////////////////////////////////////////////////////////////////////
//
// Class: View.h
////////////////////////////////////////////////////////////////////
// Purpose: To serve as an abstract class defining the common
//protocol to be followed by all view classes.
//
// Author : Bheeman
//
// Date   : January 17,  1992.
//
// Synopsis: The 'View' class is intended to be an abstract class,
//           derived from Iface class.
//
//           The constructor simply calls the parent class's
//           constructor to initialize the name.
//
//           The class defines a pure virtual function update to be
//           implemented by derived classes for implementing the the
//           corresponding views.
//
//           There is no implementation file.
////////////////////////////////////////////////////////////////////
#ifndef VIEW_H
#define VIEW_H
#include "Iface.h"
class Model;
class View : public Iface
{
 protected:

          View (char *name ) : Iface ( name ) { }

 public:
           virtual void update ( Model * ) = 0;
           virtual const char *const className() { ( "view" ); }
}
#endif
```

```
////////////////////////////////////////////////////
//   Implementation of Class Frame                //
////////////////////////////////////////////////////
#include "Frame.h"
#include <assert.h>
void main ( unsigned int argc, char **argv )
{
 assert ( theFrame );
 theFrame->initialize ( &argc, argv );
 theFrame->handleEvents();
}

////////////////////////////////////////////////////////////////
// Class: Iface
//
// Purpose: To provide facilities for Widget destruction
// & Customization of user interface
//
// Author: Bheeman
//
// Date  : Nov 28, 1992
//
// Synopsis: The Iface class defines the protocol for all
//  derived classes with respect to widget destruction
//  and customization of user interface.
//
//          installDestroyHandler() function registers the callback,
//          widgetDestroyedCallback(). This callback function invokes
//          widgetDestroyed() to initialize the base widget to NULL
//
//          The derived classes must call installDestroyHandler()
//          immediately after creating the base widget.
//
//  The costructor simply calls the Root class's constructor.
//
//          The destructor removes the XmNdestroyCallback registered
//          by installDestroyHandler(), if the base widget is non-NULL,
//          so that Xt can't call the callback with a pointer to an
//          object that has already been removed
//
//          the function setDefaultResources() sets the default
//          resource for the application.  The function getResources()
```

```
//          gets the user defined resources for the application.
////////////////////////////////////////////////////////////////////////
#ifndef IFACE_H
#define IFACE_H
#include <Xm/Xm.h>
#include "Root.h"
class Iface : public Root {
 static void widgetDestroyedCallback ( Widget,
                                       XtPointer,
                                       XtPointer );

 protected:
        Iface (const char *);
        void installDestroyHandler();
        virtual void widgetDestroyed();
        void setDefaultResources ( const Widget, const String * );
        void getResources (const XtResourceList, const int );

public:
        virtual ~Iface();
        virtual void manage();
        virtual const char *const className () { return (" Iface" ); }
};
#endif

//////////////////////////////////////////////////////
//   Implementation of Class Iface                  //
//////////////////////////////////////////////////////
#include "Iface.h"
#include <assert.h>
#include <stdio.h>
////////////////////////////////
/// interface class constructor //
////////////////////////////////
Iface::Iface ( const char *name ) : Root ( name )
{
}

////////////////////////////////
/// interface class destructor //
////////////////////////////////
Iface::~Iface() {
```

```
  if( _w )
      XtRemoveCallback ( _w,
      XmNdestroyCallback,
     &Iface::widgetDestroyedCallback,
      ( XtPointer ) this );
}

void Iface::widgetDestroyedCallback ( Widget,
                                      XtPointer clientData,
                                      XtPointer )
  {
 Iface *obj = ( Iface * ) clientData;
 obj->widgetDestroyed();
}

void Iface::widgetDestroyed () {
 _w = NULL;
 }

void Iface::installDestroyHandler () {
  assert ( _w != NULL );
  XtAddCallback ( _w,
  XmNdestroyCallback,
  &Iface::widgetDestroyedCallback,
  ( XtPointer ) this );
}

void Iface::manage() {
  assert ( _w != NULL );
  assert ( XtHasCallbacks
  ( _w, XmNdestroyCallback ) == XtCallbackHasSome );
  XtManageChild ( _w );
}


void Iface::getResources ( const XtResourceList resources,
                           const int numResources )
{
  assert ( _w != NULL );
  assert ( resources != NULL );
  XtGetSubresources ( XtParent ( _w ),
                    ( XtPointer ) this,
```

```
                    _name,
                    className(),
                    resources,
                    numResources,
                    NULL,
                    0 );
}

////////////////////////////////////////
// Setting the default resources      //
////////////////////////////////////////
void Iface::setDefaultResources ( const Widget w,
                                  const String *resourceSpec )
{
 int i;
 Display *dpy = XtDisplay ( w );
 XrmDatabase rdb = NULL;
 rdb = XrmGetStringDatabase ( "");
 i = 0;
 while ( resourceSpec[i] != NULL )
 {
  char buf[1000];
  sprintf(buf, "%s%s", _name, resourceSpec[i++]);
  XrmPutLineResource ( &rdb, buf );
 }
 if ( rdb )
 {
  XrmMergeDatabases ( dpy->db, &rdb );
  dpy->db = rdb;
 }
}

////////////////////////////////////////////////////////////////////
// class   :  IconifyWin
////////////////////////////////////////////////////////////////////
// Purpose :  To iconify all MainWin objects
//
// Author  : Bheeman
//
// Date    : December28, 1992
//
// Synopsis: This class invokes iconify() member function of the
```

```
//          global object theFrame  to close the windows of the
//          application
////////////////////////////////////////////////////////////////
#ifndef ICONIFYWIN_H
#define ICONIFYWIN_H
#include "NoUndo.h"
class  IconifyWin : public NoUndo {

  protected:

          virtual void doit();       // Iconify all windows

  public:
          IconifyWin ( char *, int );
          virtual const char *const className ()
  { return ( "IconifyWin"); }
 };
#endif

////////////////////////////////////////////////////
//   Implementation of Class IconifyWin           //
////////////////////////////////////////////////////
#include "IconifyWin.h"
#include "Frame.h"
IconifyWin::IconifyWin ( char *name, int active ) :
NoUndo ( name, active )
{

}

void IconifyWin::doit()
{

 theFrame->iconify();  // Close all top-level windows

}
////////////////////////////////////////////////////////////////
// Class: Frame.h
////////////////////////////////////////////////////////////////

// Purpose: To Provide facilities for Initialization and event
//          for the applicaation
```

```
//
// Author  : Bheeman
//
// Date    : November 27, 1992
//
// Synopsis: The application instantiates this class instead of
//           calling XtApplication() or XtAppMainLoop
//
//           It maintains a pointer to X Display required by Xt
//           functions, the name of the application and class name
//           of the application
//
//           Main() is the only function that calls the two member
//           functions, initialize and handleEvents(). So, these
//           functions are declared in the protected part of this
//           class and Main() is declared as a a friend.
//           This will also facilitate derived classes to override them.
//
//           The manage() and unmanage() functions of Iface class are
//           overridden in this class.
////////////////////////////////////////////////////////////////////////
#ifndef FRAME_H
#define FRAME_H
#include "Iface.h"
class Frame : public Iface {
  friend void main ( unsigned int, char ** );
  friend class MainWin;
 private:

         // For registering and unregistering top-level Windows
          void registerWindow (MainWin *);
          void unregisterWindow ( MainWin * );
 protected:
         // For storing data structures needed by the application
          Display  *_display;
          XtAppContext _appContext;

          // Event handler functions
          virtual void initialize ( unsigned int *, char **);
          virtual void handleEvents();
          char *_frameClass;
          MainWin **_windows;
```

```
        int     _numWindows;

public:
        Frame (char * );
        virtual ~Frame();

// To manage top level Windows. Overrides functions of Iface class
        void manage();
        void unmanage();
        void iconify();

        // Functions to access the common data
        Display *display()  { return (_display); }
        XtAppContext appContext()  { return ( _appContext); }
        const char *frameClass()  { return (_frameClass); }
        Widget rootWidget() { return (_w); };
        virtual const char *const className() { return ( "Frame"); }
};
extern Frame *theFrame;
#endif
////////////////////////////////////////////////////
//   Implementation of Class Frame                //
////////////////////////////////////////////////////
#include "Frame.h"
#include "MainWin.h"
#include <assert.h>
#include <stdlib.h>
Frame *theFrame = NULL;
Frame::Frame ( char *frameClassName ) : Iface (frameClassName)
{
  theFrame = this;
  _display = NULL;
  _appContext = NULL;
  _windows = NULL;
  _numWindows = 0;
  _frameClass = strdup ( frameClassName );
}

void Frame::initialize ( unsigned int *argcp, char **argv )
{
 _w = XtAppInitialize ( &_appContext, _frameClass, NULL, 0,
                        argcp, argv, NULL, NULL, 0 );
```

```
  // Creates the base widget
 _display = XtDisplay ( _w );

 installDestroyHandler();

 XtVaSetValues ( _w,
 XmNmappedWhenManaged, FALSE,
 XmNx, DisplayWidth ( _display, 0 ) / 2,
 XmNx, DisplayHeight ( _display, 0 ) / 2,
 XmNwidth, 100,
 XmNheight, 100,
 NULL );              // Make the base widget invisible

 delete _name;
 // Reset the program name to the one passed as
 _name  = strdup ( argv[0] );         // as an argument
 XtRealizeWidget ( _w );

 for( int i = 0; i < _numWindows; i++)
 {
     _windows[i]->initialize();
     _windows[i]->manage();
  }
}

Frame::~Frame()
{
 delete _frameClass;           // delete the application class name
 delete _windows;              // delete the list of windows
 }

void Frame::handleEvents()
{
 XtAppMainLoop ( _appContext );
}

void Frame::registerWindow (MainWin *window )
{
 int i;
 MainWin **newList;
 newList = new MainWin*[_numWindows + 1];
 for(i = 0; i < _numWindows; i++)
```

```
    newList[i] = _windows[i];

  delete []_windows;
  _windows = newList;
  _windows[_numWindows] = window;

  _numWindows++;
}

void Frame::unregisterWindow ( MainWin *window)
{
 int i, index;
 MainWin **newList;
 newList = new MainWin*[_numWindows - 1];
 index = 0;
 for(i = 0; i < _numWindows; i++)
    if( _windows[i] != window )
        newList[index++] = _windows[i];

 delete []_windows;
 _windows = newList;

 _numWindows--;

}

void Frame::manage()
{

 for (int i = 0; i < _numWindows; i++)
 _windows[i]->manage();
}

void Frame::unmanage()
{

 for ( int i = 0; i < _numWindows; i++)
 _windows[i]->unmanage();

}
void Frame::iconify()
{
```

```
 for(int i = 0; i < _numWindows; i++)
     _windows[i]->iconify();

}
////////////////////////////////////////////////////////////////////
// Class : DialogMgr
//
// Purpose : To serve as an abstract class that provides a
//           dialog caching mechanism
//
// Author : Bheeman
//
// Date   : December 24, 1992
//
// Synopsis: DialogMgr class cahes dialogs for efficient dialog
//           management.
//
//           Posts dialogs requested by the derived classes
//           (specific dialog manager classes).
//
//           Handles dialog callbacks as requested by derived classes
//
//           The class uses the shell widget of the Frame class to act
//           as parent of all dialog widgets.
//
////////////////////////////////////////////////////////////////////
#ifndef DIALOGMGR_H
#define DIALOGMGR_H
#include "Iface.h"
#include "DialogData.h"
class DialogMgr : public Iface {

 private:
         Widget getDialog();

         static void destroyTmpDialogCallback ( Widget,
                                                XtPointer,
                                                XtPointer );

         static void okCallback ( Widget,
                                  XtPointer,
```

```
                                    XtPointer );

        static void cancelCallback ( Widget,
                                     XtPointer,
                                     XtPointer );

        static void helpCallback ( Widget,
                                   XtPointer,
                                    XtPointer );

        void cleanup ( Widget, DialogData*);

protected:
         virtual Widget createDialog ( Widget ) = 0;
 // called  by derived classes to
         // create new dialog

public:
         DialogMgr ( char * );
         virtual Widget post ( char *,
         void *clientData    = NULL,
         DialogCallback ok   = NULL,
         DialogCallback cancel = NULL,
         DialogCallback help   = NULL );
};
#endif


////////////////////////////////////////////////////
//   Implementation of Class DialogMgr            //
////////////////////////////////////////////////////
#include "DialogMgr.h"
#include "Frame.h"
#include <Xm/MessageB.h>
#include <assert.h>
                  // Initialize the component's name
DialogMgr::DialogMgr ( char *name ) : Iface ( name )
{

}

////////////////////////////////////////////////////////////////////
```

```
// Check for existence of base widget, if exeists, return it, else //
// create a new new dialog widget by calling the derived class's   //
// createDialog function                                           //
/////////////////////////////////////////////////////////////////////
Widget DialogMgr::getDialog()
{
 Widget newDialog = NULL;

 if( _w && !XtIsManaged ( _w ) )
       return ( _w );

 newDialog = createDialog  ( theFrame->rootWidget() );

 // Add callbacks to destroy the temporary dialog widget(newDialog) if
 // w already exists
 if ( _w )
 {
   XtAddCallback ( newDialog,
                   XmNokCallback,
                  &DialogMgr::destroyTmpDialogCallback,
                  ( XtPointer ) this );

   XtAddCallback ( newDialog,
                   XmNcancelCallback,
                   &DialogMgr::destroyTmpDialogCallback,
                   ( XtPointer ) this );
}

 else
   _w = newDialog;
   // If w does not already exist save newDialog in w to be used
   // later also
 return ( newDialog );

}

/////////////////////////////////////////////////////////
//   To destroy the temporary dialog widget            //
/////////////////////////////////////////////////////////
void DialogMgr::destroyTmpDialogCallback ( Widget w,
                                           XtPointer,
                                            XtPointer )
```

```
{
 XtDestroyWidget ( w );
}

////////////////////////////////////////////////
// To display a dialog widget on the screen    //
////////////////////////////////////////////////
Widget DialogMgr::post ( char      *text,
                         void      *clientData,
DialogCallback ok,
DialogCallback cancel,
DialogCallback help )

{

 Widget dialog = getDialog();    // get a dialog widget

 // verify if dialog exists and it is of type xmMessageBox
 //or it's subtype

 assert ( dialog );
 assert ( XtIsSubclass (dialog, xmMessageBoxWidgetClass ) );

 XmString xmstr = XmStringCreateSimple ( text );
 XtVaSetValues (dialog, XmNmessageString, xmstr, NULL );
 XmStringFree ( xmstr );

 DialogData *dd = new DialogData ( this,
 clientData,
 ok, cancel, help );

 XtAddCallback ( dialog,
 XmNokCallback,
 &DialogMgr::okCallback,
 ( XtPointer ) dd );

 XtAddCallback ( dialog,
 XmNcancelCallback,
 &DialogMgr::cancelCallback,
 ( XtPointer ) dd );

// If there is no help callback, unmanage the corresponding button
```

```
if( help )
    XtAddCallback ( dialog,
    XmNhelpCallback,
    &DialogMgr::helpCallback,
    ( XtPointer ) dd );
else
{
    Widget w = XmMessageBoxGetChild ( dialog,
   XmDIALOG_HELP_BUTTON );
    XtUnmanageChild ( w);
 }

 XtManageChild ( dialog );
 return ( dialog );
}


void DialogMgr::okCallback ( Widget w,
                             XtPointer clientData,
                             XtPointer )
{
 DialogData      *dcd = ( DialogData * ) clientData;
 DialogMgr  *obj = ( DialogMgr * ) dcd->dialogMgr();
 DialogCallback callback;

 if((callback = dcd->ok() ) != NULL )
        ( *callback ) ( dcd->clientData() );

 obj->cleanup ( w, dcd );

}

void DialogMgr::cancelCallback ( Widget w,
                                 XtPointer clientData,
                                 XtPointer )
{

 DialogData      *dcd = ( DialogData * ) clientData;
 DialogMgr  *obj = ( DialogMgr * ) dcd->dialogMgr();
 DialogCallback callback;

 if((callback = dcd->cancel() ) != NULL )
```

```
      ( *callback ) ( dcd->clientData() );

 obj->cleanup ( w, dcd );

}
void DialogMgr::helpCallback ( Widget w,
                               XtPointer clientData,
                               XtPointer )
{

 DialogData      *dcd = ( DialogData * ) clientData;
 DialogMgr  *obj = ( DialogMgr * ) dcd->dialogMgr();
 DialogCallback callback;

 if((callback = dcd->help() ) != NULL )
    ( *callback ) ( dcd->clientData() );
 obj->cleanup ( w, dcd );
}

//////////////////////////////////////////////////////
// To remove the callbacks registered and get ready //
// for the next dialog                              //
//////////////////////////////////////////////////////
void DialogMgr::cleanup ( Widget w, DialogData *dcd )
{

XtRemoveCallback ( w,
                   XmNokCallback,
                    &DialogMgr::okCallback,
                   ( XtPointer ) dcd );


XtRemoveCallback ( w,
                    XmNcancelCallback,
                    &DialogMgr::cancelCallback,
                    ( XtPointer ) dcd );

XtRemoveCallback ( w,
                    XmNhelpCallback,
                    &DialogMgr::helpCallback,
                    ( XtPointer ) dcd );
```

```
delete dcd;

}
////////////////////////////////////////////////////////////////////
// Class : DialogData
////////////////////////////////////////////////////////////////////
// Pupose     : To serve as a container for client data to be
//              passed to various callbacks by DialogMgr class.
//
// Author     : Bheeman
//
// Date       : December 23, 1992.
//
// Synopsis   : The class declares a function pointer, DialogCallBack,
//              which represents a function that has no return value and
//              expects an untyped pointer as its only argument.
//
//              The class maintains a pointer to a DialogMgr object and
//              declares several private members including a clientData
//              field.
//
//              The constructor simply initializes the private members.
//
//              Access to the private data members is provided the
//              public member functions defined for this purpose.
//
//              There is no implementation file for this class as all
//              functions are defined inline.
////////////////////////////////////////////////////////////////////
#ifndef DIALOGDATA
#define DIALOGDATA
class DialogMgr;
typedef void ( *DialogCallback ) ( void * );

 class DialogData {

  private:
         DialogMgr  *_dialogMgr;
         DialogCallback _ok;
         DialogCallback _help;
         DialogCallback _cancel;
         void *_clientData;
```

```
  public:
          DialogData  (DialogMgr *dialog,
          void      *clientData,
          DialogCallback ok,
          DialogCallback cancel,
          DialogCallback help )
{
  _dialogMgr = dialog;
  _ok        = ok;
  _help      = help;
  _cancel    = cancel;
  _clientData = clientData;

 }
 DialogMgr  *dialogMgr()  { return (_dialogMgr ); }
 DialogCallback ok() {return ( _ok ); }
 DialogCallback help() { return ( _help ); }
 DialogCallback cancel() { return ( _cancel ); }
 void           *clientData() { return ( _clientData ); }
};
#endif

////////////////////////////////////////////////////////////////////////
//  Class  : Confirm
////////////////////////////////////////////////////////////////////////
//  Purpose : To ask the user for confirmation before actually
//            executing it
//
//  Author  : Bheeman
//
//  Date    : December 24, 1992
//
//  Synopsis: The Confirm class posts a dialog to the user
//            asking for confirmation before actually
//            executing a command. A callback is called
//            if the user  confirms the command.
//
//            The class overrides the execute() member function
//            of the Command class. This function uses the
//            theQuestnDialog object to post the question.
//            If the user confirms the command the function,
```

```
//            yesCallback is invoked. This function in turn
//            calls the Command class's execute() function to
//            to execute the doit member function and handle
//            other details involved.
//
//            The member function setQuestion() lets the
//            application choose the type of question to be posted
//
//            The constructor calls the constructor of Command class
//            and sets the default question
////////////////////////////////////////////////////////////////////
#ifndef CONFIRM_H
#define CONFIRM_H
#include "Command.h"
class Confirm : public Command {

  private:
         static void yesCallback ( void * );
         char *_question;

  public:
         Confirm ( char *, int );
         void setQuestion ( char *str );
         virtual void execute();
 // overrides the member function of Command
         virtual const char *const className () { return ("Confirm" ); }
};
#endif

////////////////////////////////////////////////////
//   Implementation of Class Confirm              //
////////////////////////////////////////////////////
#include  "Confirm.h"
#include "QuestnDialog.h"
#define DEFAULTQUESTION "Do you really want to execute this command?"
Confirm::Confirm ( char *name, int active ) : Command ( name, active )
{
   _question = NULL;
   setQuestion ( DEFAULTQUESTION );
}

void Confirm::setQuestion ( char *str )
```

```
{
 delete _question;
 _question = strdup (str );
}

void Confirm::execute()
{
 theQuestnDialog->post ( _question, ( void * ) this,
                      &Confirm::yesCallback );
}

void Confirm::yesCallback ( void *clientData )
{
 Confirm *obj = (Confirm *) clientData;
 obj->Command::execute();
 // call the execute member function of
 // the base class
}
////////////////////////////////////////////////////////////////////
// Class  : Commands
////////////////////////////////////////////////////////////////////
// Purpose : To maintain a list of Command objects
//
// Author  : Bheeman
//
// Date    : November 28, 1992
//
// Synopsis: The Commands class maintains a list of pointers to Command
//           objects, and a integer variable to store the # of commmands
//           in the list
//
//           The constructor simply initializes list and # of objects to
//           0 and the destructor frees the storage holding the list.
//
//           The member function add() adds a command to the list and the
//           operator [] allows access to the individual commands in the
//           list.
//
//           The inline functions contents() and size() return the list
//           and size of list (#of comands) respectively.
////////////////////////////////////////////////////////////////////
#ifndef COMMANDS_H
```

```
#define COMMANDS_H
class Command;
class Commands {

  private:
        Command **_contents;    // List of commands
        int _numCommands;      // # of commands;

  public:
        Commands();
        virtual ~Commands();
        void add ( Command * );  // Add a command to the list
        Command **contents() { return (_contents ); }
        int size() { return (_numCommands ); }
        Command *operator[] ( int );
};
 #endif

 ////////////////////////////////////////////////////////////////////
 //  Implementation of Commands class
 ////////////////////////////////////////////////////////////////////
# include "Commands.h"
class Command;
Commands::Commands() {

  _contents = 0;
  _numCommands = 0;
}

Commands::~Commands() {
  delete []_contents;

}

void Commands::add ( Command *command ) {
  int i;
  Command **newList;

  newList = new Command*[_numCommands + 1];
  // Create a list large for one more command

  for( i = 0; i < _numCommands; i++)
```

```
        newList[i] = _contents[i];
 // copy old list to new list

  delete []_contents;
        // delete the old list
  _contents = newList;
  _contents[_numCommands] = command;
// Add the new Command
  _numCommands++;                          // Increment the # of commands
}

Command *Commands::operator[] ( int index )
{
  return (_contents[index] );
}


////////////////////////////////////////////////////////////////////////
// Class  : Command
////////////////////////////////////////////////////////////////////////
// Purpose  : To serve as an abstract base class for all the
//            command classes used by the application
//
// Author   : Bheeman
//
// Date     : November 28, 1992
//
// Synopsis : The Command class provides a public interface for all
//            command operations in the application. It supports
//            member functions for initiating actions, enabling and
//            disabling commands.
//
//            Each Command object also supports a list of commands that
//            need to be enabled or disabled when this command is executed
//
//            The constructor initializes the data members. It sets the
//            _activeList and deactiveList to NULL. The _name and _active
//            are set to values specified by the arguments.  It also
//            initializes the _hasUndo flag to TRUE because all derived
//            classes are expected to implement undo, by default.
//
//            The destructor deletes the active command list, deactive
```

```
//            command list and CmdIface objects list
////////////////////////////////////////////////////////////////////////
#ifndef COMMAND_H
#define COMMAND_H
class  Commands;
class CmdIface;
class Command {

 friend CmdIface;

 private:
      Commands *_activeList;
      Commands *_deactiveList;
      void      revert();
  // revert object to previous state
      int        _active;             // Is active flag
      int       _previouslyActive;   // previous value of active
      char      *_name;              // Name of this command
      CmdIface  **_ci;
      int       _numIfaces;

 protected:
      int       _hasUndo;
  // This command supports undo facility
      static Command *_lastCmd;
  //  pointer to last comand executed

      virtual void doit() = 0;
      virtual void undoit() = 0;

 public:
        Command ( char *, int );
        virtual ~Command();
        virtual void execute();              // execute a command
        void undo();
// to undo the command executed
        void activate();                     // Activate the object
        void deactivate();                   // Deactivate the object
        void addToActiveList( Command * );
//  Add a command to the active list
        void addToDeactiveList ( Command * );
// Add a command to the deactive list
```

```
        void registerIface ( CmdIface * );
 // register a components which uses this cmd
        int active() { return ( _active ); }   // returns active flag
        int hasUndo() { return ( _hasUndo ); } // returns hasUndo flag
        const char *const name () { return ( _name ); }
        virtual const char *const className () { return ( "Command" ); }
};
#endif

////////////////////////////////////////////////////
//   Implementation of Class Command              //
////////////////////////////////////////////////////

#include "Command.h"
#include "Commands.h"
#include "CmdIface.h"
extern Command  *theUndoCmd;
                  // External object that reverses the most recent
                  // Command when executed

Command *Command::_lastCmd = NULL;
// pointer to most recent command
Command::Command (char *name, int active ) {
  _name          = name;
  _active        = active;
  _numIfaces     = 0;
  _ci            = NULL;
  _activeList    = NULL;
  _deactiveList  = NULL;
  _hasUndo       = TRUE;

}

Command::~Command()
{
  delete _activeList;
  delete _deactiveList;
  delete _ci;

}

void Command::registerIface ( CmdIface *ci ) {
```

```
 CmdIface **newList = new CmdIface*[_numIfaces + 1];

 for ( int i = 0; i < _numIfaces; i++)
     newList[i] = _ci[i];

 delete []_ci;

 _ci = newList;

 _ci[_numIfaces] = ci;
_numIfaces++;

 if (ci)
   if(_active)
       ci->activate;
   else
       ci->deactivate();

}

////////////////////////////////////////////////////////////////
// Activate the associated interfaces,  save the value of _active
// flag in _preiviouslyActive and set the _active flag to TRUE
////////////////////////////////////////////////////////////////
void Command::activate() {

for ( int i = 0; i <_numIfaces; i++ )
  _ci[i]->activate ();

_previouslyActive = _active;
_active = TRUE;

}

////////////////////////////////////////////////////////////////
// Decativate the associated interfaces, save the value of _active
// flag in _previoslyActive and set _active flag to FALSE
////////////////////////////////////////////////////////////////
void Command::deactivate() {

 for (int i = 0; i < _numIfaces; i++ )
```

```
  _ci[i]->deactivate ();

 _previouslyActive = _active;
 _active = FALSE;

}

//////////////////////////////////////////////////////////////////
// Activate or deactivate the associated interfaces, as necessary
// to return to the previous state
//////////////////////////////////////////////////////////////////
void Command::revert()
{

 if ( _previouslyActive )
 activate();
 else
 deactivate();

}

//////////////////////////////////////////////////////////////////
// Add a command to the activeList
//////////////////////////////////////////////////////////////////
void Command::addToActiveList ( Command *command )  {

 if ( ! _activeList )
     _activeList = new Commands();
 _activeList->add ( command );

}

//////////////////////////////////////////////////////////////////
// add a command to the deactiveList
//////////////////////////////////////////////////////////////////
void Command::addToDeactiveList ( Command * command ) {

 if( !_deactiveList )
       _deactiveList = new Commands();
_deactiveList->add ( command );

}
```

```
void Command::execute()  {

 int i;

 if( !_active )
        return;        // return if the command is inactive

 doit();
 // call the dericed functions doit() to perform the action

 if ( _activeList )
  for ( i = 0; i < _activeList->size(); i++)
    ( *_activeList ) [i]->activate();
        if ( _deactiveList )

  for ( i = 0; i < _deactiveList->size(); i++)
        ( *_deactiveList ) [i]->deactivate();

}

void Command::undo() {

 int i;

 undoit();         /* call the derived class's undoit() member funtion */

 theUndoCmd->deactivate();    // deactivate the undo facility

 // revert all dependent objects to their previous states
 if ( _activeList )
  for ( i = 0; i < _activeList->size(); i++)
       ( *_activeList ) [i]->revert();

 if ( _deactiveList )
  for ( i = 0; i < _deactiveList->size(); i++)
 ( *_deactiveList ) [i]->revert();

}

////////////////////////////////////////////////////////////////
// Class  :  CmdIface
```

```
////////////////////////////////////////////////////////////////
// Purpose : To serve as an abstract class to the derived classes
//           and support features to support interaction between
//           Command objects and widgets.
//
// Author  : Bheeman
//
// Date    : December 21, 1992
//
// Synopsis: The CmdIface class activates and deactivates user
//           interface components. It also executes and associated
//           command object.
//
//           It has a _active flag that maintains the current active
//           status of the component and a pointer to a Command
//           object.
//           Each object of this class is registered with a command
//           object whereas each Command object can be associated
//           with multiple objects of this class
//
//           The constructor simply invokes the constructor of Iface
//           class
//
//           The activate() and deactivate() member functions respectively
//           activates and deactivates the widgets supported by the objects
//           of this class
//
//           The entire class is protocol is protected, and accessible
//           only to derived classes and the friend class, Command
////////////////////////////////////////////////////////////////////
#ifdef CMDIFACE_H
#define CMDIFACE_H
#include "Iface.h"
class Command;

class CmdIface : public Iface {

 friend Command;

 protected:

           Command *_command;
```

```
            static void executeCommandCallback ( Widget,
             XtPointer,
             XtPointer );
            int _active;

            CmdIface ( Command * );
            virtual void activate();
            virtual void deactivate();
};
#endif

////////////////////////////////////////////////////
//   Implementation of Class CmdIface             //
////////////////////////////////////////////////////
#include "CmdIface.h"
#include "Command.h"
CmdIface::CmdIface ( Command *cmd ) : Iface( cmd ->name() )
{
 _active = TRUE;
 _command = cmd;
 cmd->registerIface ( this );
}

void CmdIface::executeCommandCallback ( Widget,
XtPointer clientData,
XtPointer )
{
 CmdIface *obj = (CmdIface * ) clientData;
 obj->_command->execute();
}

void CmdIface::activate()
{
 if( _w )
        XtSetSensitive ( _w, TRUE );
 _active = TRUE;
}

void CmdIface::deactivate()
{
 if( _w )
        XtSetSensitive ( _w, FALSE );
```

```
 _active = FALSE;
}

//////////////////////////////////////////////////////////////////////////
//  Class : ButtonIface
//////////////////////////////////////////////////////////////////////////
//  Purpose : To provide an interface between a command and
//            the associated widget
//
//  Author  : Bheeman
//
//  Date    : December 21, 1992
//
//  Synopsis: This class is derived from CmdIface class.
//
//            The constructor creates a Motif XmPushButton widget and
//   registers the CmdIface::executeCallback()
//   function to be called when the user selects
//   the menu item.
//////////////////////////////////////////////////////////////////////////
#ifndef BUTTONIFACE_H
#define BUTTONIFACE_H
#include "CmdIface.h"
class ButtonIface : public CmdIface {

 public:
        ButtonIface ( Widget, Command *);

};
#endif

æ//////////////////////////////////////////////////
//   Implementation of Class ButtonIface          //
//////////////////////////////////////////////////
#include "ButtonIface.h"
#include <Xm/PushB.h>
ButtonIface::ButtonIface ( Widget parent,
                           Command *cmd ) : CmdIface ( cmd )
{
 _w = XtCreateWidget ( _name,
                           xmPushButtonWidgetClass,
```

```
                            parent,
                            NULL, 0 );
 installDestroyHandler();

 if( _active )
 // activate or deactivate the widget as per status flag
       activate();
 else
       deactivate();

 XtAddCallback ( _w,
                 XmNactivateCallback,
                 &CmdIface::executeCommandCallback,
                 (XtPointer) this );
}
```

APPENDIX D

C++ CLASS DEFINITIONS AND IMPLEMENTATION OF GUI

```
////////////////////////////////////////////////////////////////////
//
// Class: Model
////////////////////////////////////////////////////////////////////
// Purpose: To obtain and maintain the data needed to be displayed
//          by different View objects.
//
// Author : Bheeman
//
// Date   : January 17,  1992.
//
// Synopsis: The 'model' class responds to the messages passed by
//           Controller Class and call the appropriate member function
//           interact with the DBMS in order to get the desired data
//           list. Then it calls the updateView member funtion to update
//           the corresponding view.
//
//           The constructor initializes the the private data members
//
//           The member function updateViews() calls the update()
//           member function of the appripriate view object
//
////////////////////////////////////////////////////////////////////
#ifndef MODEL_H
#define MODEL_H
#include "ViewT1.h"
#include "ViewT2.h"
#include "ViewT3.h"
#include "ViewT4.h"
#include "ViewB1.h"
#include "ViewB2.h"
class Controller;
```

```
class Model {
 friend Controller;
 friend ViewT1;
 friend ViewT2;
 friend ViewT3;
 friend ViewT4;
 friend ViewB1;
 friend ViewB2;

private:
    int _listCount;
    XmString _outList[80];
    char _outText[1000];
    char _inText[500];
// Maximum 1000 character input string from textWindow
    char _inList[120];
// One string at a time is selected from listWindow
    char _prevWin[15];
// window in which user selected an item or input data
    char _inWin[15];
    ViewT1 *_viewT1;
// the window to be used for next display will be based on
    ViewT2 *_viewT2;
// the _prevWin
    ViewT3 *_viewT3;
    ViewT4 *_viewT4;
    ViewB1 *_viewB1;
    ViewB2 *_viewB2;
    void  updateViewT1();
    void  updateViewT2();
    void  updateViewT3();
    void  updateViewT4();
    void  updateViewB1();
    void  updateViewB2();
    void  getClasses();
    void  getInstances();
    void  getAttributes();
    void  getUserRel();
    void  getGenRel();
    void  queryDBMS();
    void  updateDBMS();
    void  identifyView();
```

```
 public:
    Model(ViewT1 *viewT1, ViewT2 *viewT2, ViewT3 *viewT3,
 ViewT4 *viewT4, ViewB1 *viewB1, ViewB2 *viewB2);
};
#endif
æ//////////////////////////////////////////////////
//   Implementation of Class Model                //
//////////////////////////////////////////////////
#include "Model.h"
#include <stdio.h>
Model::Model(ViewT1 *viewT1, ViewT2 *viewT2, ViewT3 *viewT3,
ViewT4 *viewT4, ViewB1 *viewB1, ViewB2 *viewB2)
{
  int i;

  _viewT1 = viewT1;
  _viewT2 = viewT2;
  _viewT3 = viewT3;
  _viewT4 = viewT4;
  _viewB1 = viewB1;
  _viewB2 = viewB2;
  strcpy(_prevWin, "\0");
  strcpy(_inList, "\0");
}

////////////////////////////////////////////////////////////////
// Inform ViewT1 that the data is ready
////////////////////////////////////////////////////////////////
void Model::updateViewT1()
 {
 _viewT1->update(this);
 strcpy(_prevWin, "T1");
}

////////////////////////////////////////////////////////////////
// Inform ViewT1 that the data is ready
////////////////////////////////////////////////////////////////
void Model::updateViewT2()
{
 _viewT2->update(this);
 strcpy(_prevWin, "T2");
```

```
}

//////////////////////////////////////////////////////////////////
// Inform ViewT1 that the data is ready
//////////////////////////////////////////////////////////////////
void Model::updateViewT3()
{
 _viewT3->update(this);
 strcpy(_prevWin, "T3");
}

//////////////////////////////////////////////////////////////////
// Inform ViewT1 that the data is ready
//////////////////////////////////////////////////////////////////
void Model::updateViewT4()
{
 _viewT4->update(this);
 strcpy(_prevWin, "T4");
}

//////////////////////////////////////////////////////////////////
// Inform ViewT1 that the data is ready
//////////////////////////////////////////////////////////////////
void Model::updateViewB2()
{
 _viewB2->update(this);
 strcpy(_prevWin, "B2");
}


//////////////////////////////////////////////////////////////////
// For obtaining all classes in the Database and display in
// Window B1
//////////////////////////////////////////////////////////////////
void Model::getClasses() {
 int i;

 char s[80];

 for(i = 0; i < 50; i++)
 {
   sprintf(s, "%s-%d\n", "TEST-CLASS", i);
```

```
   strcat(_outText, s);

 }

  _listCount = i;

 updateViewB2();
}

//////////////////////////////////////////////////////////////
// Get Instances for a given classs
//////////////////////////////////////////////////////////////
void Model::getInstances() {
 int i;
 char s[80];

 for(i = 0; i < 10; i++)
 {
sprintf(s, "%s-%d", "TEST-INSTANCE", i);
   _outList[i] = XmStringCreateSimple(s);
 }
  _listCount = i;

 updateViewT1();

}

//////////////////////////////////////////////////////////////
// Get Attribute names of a given instance type
//////////////////////////////////////////////////////////////
void Model::getAttributes() {

 int i;
 char s[80];

 for(i = 0; i < 10; i++)
 {
sprintf(s, "%s-%d for %s", "TEST-ATTRIBUTE", i, _inList);
   _outList[i] = XmStringCreateSimple(s);
 }
  _listCount = i;
```

```
 updateViewT2();
}

//////////////////////////////////////////////////////////////
// Get user defined relationships for a given instance type
//////////////////////////////////////////////////////////////

void Model::getUserRel() {

 int i;
 char s[80];

 for(i = 0; i < 10; i++)
 {
sprintf(s, "%s-%d for %s", "TEST-RELATIONSHIP", i, _inList);
   _outList[i] = XmStringCreateSimple(s);
 }
  _listCount = i;

 updateViewT2();
}

//////////////////////////////////////////////////////////////
// Get Generic relationships for a given instance type
//////////////////////////////////////////////////////////////
void  Model::getGenRel() {

}

//////////////////////////////////////////////////////////////
// Send user a query to DBMS
//////////////////////////////////////////////////////////////
void Model::queryDBMS()
{
}

//////////////////////////////////////////////////////////////
// Send update request to DBMS
//////////////////////////////////////////////////////////////
void Model::updateDBMS()
{
}
```

```
////////////////////////////////////////////////////////////
// Identify the next window to be used for display based on
// the window name stored in _prevWin
////////////////////////////////////////////////////////////
void Model::identifyView()
{

 if(strcmp(_prevWin, "T1") == 0)
     updateViewT2();
 else if(strcmp(_prevWin, "T2") == 0)
     updateViewT3();
 else if(strcmp(_prevWin, "T3") == 0)
     updateViewT1();
}

////////////////////////////////////////////////////
//Class ViewT1                                    //
////////////////////////////////////////////////////

#ifndef VIEWT1_H
#define VIEWT1_H
#include "View.h"
class ViewT1 : public View {

 private:
      Widget _scrollListT1;
      char *_inputstr;
      static void getValueCallback(Widget, XtPointer,
                                   XtPointer);

 public:

      ViewT1 ( char *name, Widget parent);
      void update ( Model *model);
      virtual const char *const className()
  { return ( "viewT1" ); }
};
#endif

////////////////////////////////////////////////////
//   Implementation of Class ViewT1               //
```

```
////////////////////////////////////////////////////

#include "View.h"
#include "ViewT1.h"
#include "Controller.h"
#include "Model.h"
#include <Xm/List.h>
#include <Xm/Xm.h>
#include <iostream.h>
extern Controller *theController;
ViewT1::ViewT1(char *name,  Widget parent) : View( name )
{
  _scrollListT1 = XmCreateScrolledList( parent,
  "scrollListT1",
  NULL,
  0);

_w = XtParent(_scrollListT1);
installDestroyHandler();


XtVaSetValues ( _w, XmNscrollBarPlacement, XmBOTTOM_LEFT,
               NULL);

XtVaSetValues ( _scrollListT1, XmNvisibleItemCount, 18,
                 XmNscrollBarDisplayPolicy, XmSTATIC,
                 XmNselectionPolicy, XmSINGLE_SELECT,
                 NULL );

XtAddCallback ( _scrollListT1,
                 XmNsingleSelectionCallback,
                 &ViewT1::getValueCallback,
                 (XtPointer) this );

XtManageChild(_scrollListT1);
XtManageChild(_w);

}

////////////////////////////////////////////////////////////
// This function is invoked my Model class for updating the
// view buffer
```

```
////////////////////////////////////////////////////////
void ViewT1::update( Model *model ) {

  XtVaSetValues(_scrollListT1,
                XmNitems, model->_outList,
                XmNitemCount, model->_listCount,
                NULL );
// Access data from model

}

////////////////////////////////////////////////////////
// callback function to pass the string input by user to
// Model Class
////////////////////////////////////////////////////////

void ViewT1::getValueCallback(Widget w, XtPointer clientData,
 XtPointer callData)
{
 XmString xmstr;
 int flag;
 ViewT1 *obj = (ViewT1 *) clientData;
 XmListCallbackStruct *cb = (XmListCallbackStruct *) callData;
 xmstr = cb->item;
 flag = XmStringGetLtoR(xmstr, XmSTRING_DEFAULT_CHARSET,
   &obj->_inputstr);
 theController->registerFromViewT1(obj->_inputstr);
}

////////////////////////////////////////////////////////
//Class ViewT2                                        //
////////////////////////////////////////////////////////
#ifndef VIEWT2_H
#define VIEWT2_H
#include "View.h"
class ViewT2 : public View {

 private:
        Widget _scrollListT2;
        char *_inputstr;
static void getValueCallback(Widget, XtPointer, XtPointer);
```

```
 public:
        ViewT2 ( char *name, Widget parent);
        void update ( Model *model);
        virtual const char *const className()
{ return ( "viewT2" ); }
};
#endif

////////////////////////////////////////////////////
//   Implementation of Class ViewT2               //
////////////////////////////////////////////////////
#include "View.h"
#include "ViewT2.h"
#include "Controller.h"
#include "Model.h"
#include <Xm/List.h>
#include <Xm/Xm.h>
#include <iostream.h>
extern Controller *theController;
ViewT2::ViewT2(char *name,  Widget parent) : View( name )
{
   _scrollListT2 = XmCreateScrolledList( parent,
   "scrollListT2",
NULL,
    0);

_w = XtParent(_scrollListT2);
installDestroyHandler();


XtVaSetValues ( _w, XmNscrollBarPlacement, XmBOTTOM_LEFT,
               NULL);

XtVaSetValues ( _scrollListT2, XmNvisibleItemCount, 18,
               XmNscrollBarDisplayPolicy, XmSTATIC,
               XmNselectionPolicy, XmSINGLE_SELECT,
               NULL );

XtAddCallback ( _scrollListT2,
                  XmNsingleSelectionCallback,
                   &ViewT2::getValueCallback,
                  (XtPointer) this );
```

```
XtManageChild(_scrollListT2);
XtManageChild(_w);
}

////////////////////////////////////////////////////////////
// This function is invoked my Model class for updating the
// view buffer
////////////////////////////////////////////////////////////
void ViewT2::update( Model *model ) {

  XtVaSetValues(_scrollListT2,
                XmNitems, model->_outList,
                XmNitemCount, model->_listCount,
                NULL );
// Access data from model

}

////////////////////////////////////////////////////////////
// callback function to pass the string input by user to
// Model Class
////////////////////////////////////////////////////////////

void ViewT2::getValueCallback(Widget w, XtPointer clientData,
 XtPointer callData)
{
 XmString xmstr;
 int flag;

 ViewT2 *obj = (ViewT2 *) clientData;
 XmListCallbackStruct *cb = (XmListCallbackStruct *) callData;
 xmstr = cb->item;
 flag = XmStringGetLtoR(xmstr, XmSTRING_DEFAULT_CHARSET,
   &obj->_inputstr);
 theController->registerFromViewT2(obj->_inputstr);
}

////////////////////////////////////////////////////
//Class ViewT3                                    //
////////////////////////////////////////////////////
#ifndef VIEWT3_H
```

```
#define VIEWT3_H
#include "View.h"
class ViewT3 : public View {

 private:
        Widget _scrollListT3;
        char *_inputstr;
        static void getValueCallback(Widget, XtPointer,
 XtPointer);

 public:
        ViewT3 ( char *name, Widget parent);
        void update ( Model *model);
        virtual const char *const className()
{ return ( "viewT3" ); }
};
#endif

//////////////////////////////////////////////////
//   Implementation of Class ViewT3             //
//////////////////////////////////////////////////
#include "View.h"
#include "ViewT3.h"
#include "Controller.h"
#include "Model.h"
#include <Xm/List.h>
#include <Xm/Xm.h>
#include <iostream.h>
extern Controller *theController;
ViewT3::ViewT3(char *name,  Widget parent) : View( name )
{
  _scrollListT3 = XmCreateScrolledList( parent,
   "scrollListT3",
NULL,
   0);

_w = XtParent(_scrollListT3);
installDestroyHandler();



XtVaSetValues ( _w, XmNscrollBarPlacement, XmBOTTOM_LEFT,
```

```
                NULL);

 XtVaSetValues ( _scrollListT3, XmNvisibleItemCount, 18,
                 XmNscrollBarDisplayPolicy, XmSTATIC,
                XmNselectionPolicy, XmSINGLE_SELECT,
                NULL );

 XtAddCallback ( _scrollListT3,
                XmNsingleSelectionCallback,
                &ViewT3::getValueCallback,
                (XtPointer) this );

 XtManageChild(_scrollListT3);
 XtManageChild(_w);

 }

 ////////////////////////////////////////////////////////////
 // This function is invoked my Model class for updating the
 // view buffer
 ////////////////////////////////////////////////////////////
 void ViewT3::update( Model *model ) {
 int i;

   XtVaSetValues(_scrollListT3,
              XmNitems, model->_outList,
              XmNitemCount, model->_listCount,
              NULL );
 // Access data from model

 }

 ////////////////////////////////////////////////////////////
 // callback function to pass the string input by user to
 // Model Class
 ////////////////////////////////////////////////////////////

 void ViewT3::getValueCallback(Widget w, XtPointer clientData,
  XtPointer callData)
 {
  XmString xmstr;
  int flag;
```

```
 ViewT3 *obj = (ViewT3 *) clientData;
 XmListCallbackStruct *cb = (XmListCallbackStruct *) callData;
 xmstr = cb->item;
 flag = XmStringGetLtoR(xmstr, XmSTRING_DEFAULT_CHARSET,
   &obj->_inputstr);
 theController->registerFromViewT3(obj->_inputstr);
}

////////////////////////////////////////////////////
//Class ViewT4                                    //
////////////////////////////////////////////////////
#ifndef VIEWT4_H
#define VIEWT4_H
#include "View.h"
class ViewT4 : public View {

 private:
       Widget _scrollTextT4;
       char _inText[1000];

 public:
       ViewT4 ( char *name, Widget parent);
       void update ( Model *model);
       virtual const char *const className()
   { return ( "viewT4" ); }
};
#endif

////////////////////////////////////////////////////
//   Implementation of Class ViewT4               //
////////////////////////////////////////////////////
#include "View.h"
#include "ViewT4.h"
#include "Controller.h"
#include "Model.h"
#include <Xm/Xm.h>
#include <Xm/Text.h>
#include <iostream.h>
ViewT4::ViewT4(char *name,  Widget parent) : View( name )
{
 _scrollTextT4 = XmCreateScrolledText( parent,
```

```
 "scrollTextT4",
NULL,
 0);

_w = XtParent(_scrollTextT4);
installDestroyHandler();
XtVaSetValues ( _w, XmNscrollBarPlacement, XmBOTTOM_LEFT,
                NULL);
XtVaSetValues(_scrollTextT4, XmNeditable, TRUE,
              XmNeditMode, XmMULTI_LINE_EDIT,
              XmNmaxLength, 1000,
              NULL);

XtManageChild(_scrollTextT4);
XtManageChild(_w);

}

////////////////////////////////////////////////////////////
// This function is invoked my Model class for updating the
// view buffer
////////////////////////////////////////////////////////////
void ViewT4::update( Model *model ) {
 int i;


 XtVaSetValues(_scrollTextT4,
              XmNvalue, model->_outText,
              NULL );
// Access data from model
}


////////////////////////////////////////////////////
// Class Controller                               //
////////////////////////////////////////////////////
#ifndef CONTROLLER_H
#define CONTROLLER_H
#include <Xm/Xm.h>
class Model;
class Controller {
 private:
```

```
        Model *_model;
        char *_action, *_selection,  *_win;
        char *_inValue[80];
        char * className;
        char *_newValue;
        char *_instanceName;
        char *_propName;
        char *_propValue;
        char *_queryStatement;
        int _maxLen;

 public:
        Controller( Model *model);
        void registerAdd();
        void registerModify();
        void re isterDelete();
        void registerPrint();
        void registerClasses();
        void registerInstances();
        void registerAttributes();
        void registerUserDefRel();
        void registerGenericRel();
        void registerFromViewT1(const char *str);
        void registerFromViewT2(const char *str);
        void registerFromViewT3(const char *str);
        void registerFromViewT4(const char *str);
         void registerFromViewB1(const char *str);
         //void registerFromViewB2(const char *str);
         void proceed();
         void cancel();
};
extern Controller *theController;
#endif

////////////////////////////////////////////////////
//   Implementation of Class Controller           //
////////////////////////////////////////////////////
#include "Controller.h"
#include "Model.h"
#include "./libuniv/Frame.h"
#include <iostream.h>
#include <stdio.h>
```

```
Controller *theController = NULL;

Controller::Controller( Model *model)
{
 int i;
 theController = this;
 _model = model;
 _action = NULL;       // action selected from menu
 _selection = NULL;    // to hold item type selected from menu
 _win = NULL;          // the recent window that was activated
 _maxLen = 20;         // Max length of action/selection string
}

void Controller::registerAdd()
{
 if(_action)
      delete _action;
 _action = new char(_maxLen);
 _action = "add";
}

void Controller::registerModify()
{
 if(_action)
             delete _action;
 _action = new char[_maxLen];
 _action = "modify";
}
void Controller::registerDelete()
{
 if(_action)
         delete _action;
 _action = new char(_maxLen);
 _action = "delete";
}
void Controller::registerPrint()
{
 if(_action)
         delete _action;
 _action = new char[_maxLen];
 _action = "print";
}
```

```
void Controller::registerClasses()
{
 if(_selection)
        delete _selection;
 _selection = new char[_maxLen];
 _selection = "classes";
}
void Controller::registerInstances()
{
 if(_selection)
        delete _selection;
 _selection = new char[_maxLen];
 _selection = "instances";
}
void Controller::registerAttributes()
{
 if(_selection)
        delete _selection;
 _selection = new char(_maxLen);
 _selection = "attributes";
}
void Controller::registerUserDefRel()
{
 if(_selection)
        delete _selection;
 _selection = new char(_maxLen);
_selection = "userDefRel";

}
void Controller::registerGenericRel()
{
 if(_selection)
        delete _selection;
 _selection = new char(_maxLen);
 _selection = "genericRel";

}


//////////////////////////////////////////////////////
// Function to get input  string from listwindowT1
```

```
// passed from ViewT1 and call store it in the
// member _entity
////////////////////////////////////////////////////
void Controller::registerFromViewT1( const char *str)
{
 if(_instanceName)
       delete _instanceName;
 _instanceName = strdup((char *) str);
 if(_win)
 delete _win;
 _win = new char(3);
 _win = "T1";
  //_model->registerListInput( _selValue, _win);
}
////////////////////////////////////////////////////
// Function to get input  string from listwindowT2
// passed from ViewT2 and call store it in the
// member _entity
////////////////////////////////////////////////////
void Controller::registerFromViewT2( const char *str)
{
 if(_propName)
         delete _propName;
 _propName = strdup((char *)str);

 if(_win)
         delete _win;
 _win = new char(3);
 _win = "T2";
 // _model->registerListInput( _selValue, _win);
}
////////////////////////////////////////////////////
// Function to get input  string from listwindowT3
// passed from ViewT3 and call store it in the
// member _entity
////////////////////////////////////////////////////
void Controller::registerFromViewT3( const char *str)
{
 if(_propValue)
         delete _propValue;
 _propValue = strdup( (char *)str);
```

```
 if(_win)
         delete _win;
 _win = new char(3);
 _win = "T3";
 // _model->registerListInput( _selValue, _win);
}
////////////////////////////////////////////////////////
// Function to get input an string from Text WindowT4
// passed from ViewB1 and store it in the member _entity
////////////////////////////////////////////////////////
void Controller::registerFromViewT4( const char *str)
{
 if(_newValue)
         delete _newValue;
 _newValue = strdup((char *)str);

 if(_win)
        delete _win;
  _win = new char(3);
 _win = "T4";
  cout << _newValue << endl;
 //   _model->registerListInput( _selValue, _win);
}

////////////////////////////////////////////////////////
// Function to get input an string from TextwindowB1
// passed from ViewB1 and store it in the member _entity
////////////////////////////////////////////////////////
void Controller::registerFromViewB1( const char *str)
{
 _queryStatement = strdup( (char *)str);

 if(_win)
        delete _win;
 _win = new char(3);
 _win = "B1";

  cout << _queryStatement << endl;
 _model->queryDBMS();
//    _model->registerListInput( _selValue, _win);
}
```

```
//////////////////////////////////////////////////////
// This method makes the decision based on SELECTION,
// ACTION and in put from  Views, if any
//////////////////////////////////////////////////////
void Controller::proceed()
{
 if((strcmp(_action, "print") == 0)
 && (strcmp(_selection, "classes") == 0))
     _model->getClasses();
 else if((strcmp(_action, "print") == 0)
   && (strcmp(_selection, "instances") == 0))
       _model->getInstances();
 else if((strcmp(_action, "print") == 0)
   && (strcmp(_selection, "attributes") == 0))
        _model->getAttributes();

}
//////////////////////////////////////////////////////
// Cancels the intended action by the user
//////////////////////////////////////////////////////
void Controller::cancel()
{
 if(_action)
      delete _action;
 if(_selection)
      delete _selection;
 if(_className)
      delete _className;
 if(_win)
    delete _win;
 if(_instanceName)
      delete _instanceName;
 if(_propName)
      delete _propName;
 if(_propValue)
      delete _propValue;

 _action = NULL;
 _selection = NULL;
 _className = NULL;
 _instanceName = NULL;
 _propName = NULL;
```

```
 _propValue = NULL;
 _win = NULL;
}

////////////////////////////////////////////////////
//Class Add                                       //
////////////////////////////////////////////////////
#ifndef ADD_H
#define ADD_H
#include "./libuniv/Command.h"
class Add : public Command {

 protected:

      virtual void doit();
      virtual void undoit();

 public:
      Add ( char *, int );
      virtual const char *const className () { return ( "Add" ); }

};
#endif

////////////////////////////////////////////////////
//   Implementation of Class Add                  //
////////////////////////////////////////////////////
#include "Add.h"
#include "./libuniv/Frame.h"
#include <iostream.h>
Add::Add ( char *name, int active ) : Command ( name, active )
{
//
}

void Add::doit()
{

cout << name() << ":" << "doit\n" << flush;
}

void Add::undoit()
```

```
{

 cout << name() << ":" << "undoit\n" << flush;
}

//////////////////////////////////////////////////////
//Class Delete                                      //
//////////////////////////////////////////////////////
#ifndef DELETE_H
#define DELETE_H
#include "./libuniv/Command.h"

class Delete : public Command {

 protected:
        virtual void doit();
        virtual void undoit();

 public:
        Delete ( char *, int );
        virtual const char *const className ()
{ return ( "Delete" ); }
};
#endif

//////////////////////////////////////////////////////
//   Implementation of Class Delete                 //
//////////////////////////////////////////////////////
#include "Delete.h"
#include "./libuniv/Frame.h"
#include <iostream.h>

Delete::Delete ( char *name, int active ) :
 Command ( name, active )
{

}

void Delete::doit()
{

    cout << name() << ":" << "doit\n" << flush;
```

```
}

void Delete::undoit()
{

 cout << name() << ":" << "undoit\n" << flush;

}


//////////////////////////////////////////////////////
//Class Modify                                      //
//////////////////////////////////////////////////////
#ifndef MODIFY_H
#define MODIFY_H
#include "./libuniv/Command.h"
class Modify : public Command {

 protected:
         virtual void doit();
         virtual void undoit();

 public:
         Modify ( char *, int );
         virtual const char *const className ()
 { return ( "Modify" ); }

};
#endif

//////////////////////////////////////////////////////
//   Implementation of Class Modify                 //
//////////////////////////////////////////////////////
#include "Modify.h"
#include "./libuniv/Frame.h"
#include <iostream.h>
Modify::Modify ( char *name, int active ) :
                         Command ( name, active )
{

}
```

```
void Modify::doit()
{

 cout << name() << ":" << "doit\n" << flush;

}

void Modify::undoit()
{

 cout << name() << ":" << "undoit\n" << flush;

}


æ
////////////////////////////////////////////////////
//Class Print                                     //
////////////////////////////////////////////////////

#ifndef PRINT_H
#define PRINT_H
#include "./libuniv/Command.h"
class Print : public Command {

 protected:
          virtual void doit();
          virtual void undoit();

 public:

          Print ( char *, int );
          virtual const char *const className ()
  { return ( "Print" ); }

};
#endif

////////////////////////////////////////////////////
//   Implementation of Class Print                //
////////////////////////////////////////////////////
```

```
#include "Print.h"
#include "./libuniv/Frame.h"
#include <iostream.h>
#include "Controller.h"
extern Controller *theController;
Print::Print ( char *name, int active ) :
  Command ( name, active )
{

}

void Print::doit()
{
 theController->registerPrint();

}

void Print::undoit()
{

 cout << name() << ":" << "undoit\n" << flush;

}

//////////////////////////////////////
//  Class Attributes
//////////////////////////////////////
#ifndef ATTRIBUTES_H
#define ATTRIBUTES_H
#include "./libuniv/Command.h"
class Attributes : public Command {

 protected:
        virtual void doit();
        virtual void undoit();

 public:
        Attributes ( char *, int );
        virtual const char *const className ()
 { return ("Attributes" );}
};
#endif
```

```
////////////////////////////////////////////////////
//   Implementation of Class Attributes           //
////////////////////////////////////////////////////
#include "Attributes.h"
#include "./libuniv/Frame.h"
#include <iostream.h>
#include "Controller.h"
extern Controller *theController;
 Attributes::Attributes ( char *name, int active ):
Command ( name, active )
{

}

void Attributes::doit()
{

theController->registerAttributes();


void Attributes::undoit()
{

 cout << name() << ":" << "undoit\n" << flush;

}

////////////////////////////////////////////////////
// Class Classes                                  //
////////////////////////////////////////////////////
#ifndef CLASSES_H
#define CLASSES_H
#include "./libuniv/Command.h"
class Classes : public Command {

 protected:
          virtual void doit();
          virtual void undoit();

 public:
          Classes ( char *, int );
```

```
        virtual const char *const className ()
  { return ( "Classes" ); }
};
#endif

//////////////////////////////////////////////////////
//   Implementation of Class Classes                //
//////////////////////////////////////////////////////
#include "Classes.h"
#include "./libuniv/Frame.h"
#include <iostream.h>
#include "Controller.h"
extern Controller *theController;
 Classes::Classes ( char *name, int active ) :
 Command ( name, active )
{

}

void Classes::doit()
{

theController->registerClasses();

}

void Classes::undoit()
{
 //cout << name() << ":" << "undoit\n" << flush;
}

//////////////////////////////////////////////////////
//Class Instances                                   //
//////////////////////////////////////////////////////
#ifndef INSTANCES_H
#define INSTANCES_H
#include "./libuniv/Command.h"
class Instances : public Command {

 protected:
         virtual void doit();
         virtual void undoit();
```

```
 public:
          Instances ( char *, int );
  virtual const char *const className ()
  { return ( "Instances" ); }
};
#endif

////////////////////////////////////////////////////
//   Implementation of Class Instances            //
////////////////////////////////////////////////////
#include "Instances.h"
#include "./libuniv/Frame.h"
#include <iostream.h>
#include "Controller.h"
extern Controller *theController;
Instances::Instances ( char *name, int active ) :
  Command ( name, active )
{

}

void Instances::doit()
{
 theController->registerInstances();

}

void Instances::undoit()
{

 cout << name() << ":" << "undoit\n" << flush;

}

////////////////////////////////////////////////////
//Class UserDefRel                                //
////////////////////////////////////////////////////
#ifndef USERDEFREL_H
#define USERDEFREL_H
#include "./libuniv/Command.h"
class UserDefRel : public Command {
```

```
 protected:
          virtual void doit();
          virtual void undoit();

 public:
          UserDefRel ( char *, int );
 virtual const char *const className ()
  { return ( "UserDefRel" ); }
};
#endif

////////////////////////////////////////////////////
//   Implementation of Class UserDefRel           //
////////////////////////////////////////////////////
#include "UserDefRel.h"
#include "./libuniv/Frame.h"
#include <iostream.h>
#include "Controller.h"
extern Controller *theController;
UserDefRel::UserDefRel ( char *name, int active ) :
Command ( name, active )
{

}

void UserDefRel::doit()
{

 theController->registerUserDefRel();
}

void UserDefRel::undoit()
{

 cout << name() << ":" << "undoit\n" << flush;

}


////////////////////////////////////////////////////
//Class  GenericRel                               //
```

```
//////////////////////////////////////////////////

#ifndef GENERICREL_H
#define GENERICREL_H
#include "./libuniv/Command.h"
class GenericRel : public Command {

 protected:
        virtual void doit();
        virtual void undoit();

 public:
        GenericRel ( char *, int );
        virtual const char *const className ()
 { return ( "GenericRel" ); }

};
#endif

//////////////////////////////////////////////////
//   Implementation of Class GenericRel         //
//////////////////////////////////////////////////
#include "GenericRel.h"
#include "./libuniv/Frame.h"
#include <iostream.h>
#include "Controller.h"
extern Controller *theController;
GenericRel::GenericRel ( char *name, int active ) :
Command ( name, active )
{

}

void GenericRel::doit()
{

 theController->registerGenericRel();
}

void GenericRel::undoit()
{
```

```
 cout << name() << ":" << "undoit\n" << flush;

}

//////////////////////////////////////////////////////
//Class Go                                          //
//////////////////////////////////////////////////////
#ifndef GO_H
#define GO_H
#include "./libuniv/Command.h"
class Go : public Command {

 protected:
         virtual void doit();
         virtual void undoit();

 public:
         Go ( char *, int );
         virtual const char *const className ()
 { return ( "Go" ); }

};
#endif

//////////////////////////////////////////////////////
//   Implementation of Class Go                     //
//////////////////////////////////////////////////////

#include "Go.h"
#include "./libuniv/Frame.h"
#include <iostream.h>
#include "Controller.h"
extern Controller *theController;
Go::Go ( char *name, int active ) :
  Command ( name, active )
{

}

void Go::doit()
{
```

```
 theController->proceed();
}

void Go::undoit()
{

 cout << name() << ":" << "undoit\n" << flush;

}

////////////////////////////////////////////////////
//Class Cancel                                    //
////////////////////////////////////////////////////
#ifndef CANCEL_H
#define CANCEL_H
#include "./libuniv/Command.h"
class Cancel : public Command {

 protected:
        virtual void doit();
        virtual void undoit();

 public:
        Cancel ( char *, int );
        virtual const char *const className ()
 { return ( "Cancel" ); }
};
#endif

////////////////////////////////////////////////////
//   Implementation of Class Cancel               //
////////////////////////////////////////////////////
#include "Cancel.h"
#include "./libuniv/Frame.h"
#include <iostream.h>
#include "Controller.h"
extern Controller *theController;
Cancel::Cancel ( char *name, int active ) :
Command ( name, active )
{

}
```

```
void Cancel::doit()
{

 theController->cancel();
}

void Cancel::undoit()
{

 cout << name() << ":" << "undoit\n" << flush;

}

////////////////////////////////////////////////////////
// Class  :  MenuOpt
////////////////////////////////////////////////////////
// Purpose  : To declare the menu options for University
//            database
//
// Author   : Bheeman
//
// Date     : January 10, 1992
//
// Synopsis : The class MenuOPt declares the three Command
//            objects _manage, _iconify and _quit to to
//            appear in all menus. declares the Command
//            objects used by University database
//
//            The constructor creates the above Commands
//            objects and specify dependancies between the
//            commands
//
//            The destructor destroy Command objects created
//            by the constructor
//
//            Finally, it declares a global MenuOpt object
//            to be available for all classes
////////////////////////////////////////////////////////
#ifndef MENUOPT_H
#define MENUOPT_H
#include "./libuniv/Frame.h"
```

```
class Command;

class MenuOpt : public Frame {

 protected:
          Command *_manage;
          Command *_iconify;
          Command *_quit;
          Command *_add;
          Command *_print;
          Command *_modify;
          Command *_delete;
          Command *_classes;
          Command  *_instances;
          Command  *_attributes;
          Command  *_userDefRel;
          Command  *_genericRel;
          Command *_go;
          Command *_cancel;

  public:
          MenuOpt ( char *);
          virtual ~MenuOpt();
          Command *manageCmd() { return ( _manage ); }
          Command *iconifyCmd() { return ( _iconify ); }
          Command *quitCmd() { return ( _quit ); }
          Command *addCmd() { return ( _add ); }
          Command *printCmd() { return ( _print ); }
          Command *modifyCmd() { return ( _modify ); }
          Command *deleteCmd() { return ( _delete  ); }
          Command *classesCmd() { return (_classes); }
          Command *instancesCmd() { return (_instances ); }
          Command *attributesCmd() { return ( _attributes ); }
          Command *userDefRelCmd() { return ( _userDefRel ); }
          Command *genericRelCmd() { return ( _genericRel ); }
          Command *goCmd() { return ( _go ); }
          Command *cancelCmd() { return ( _cancel ); }
          virtual const char *const className()
    { return ("MenuOpt"); }

  };
```

```
extern MenuOpt *theMenuOpt;
#endif

////////////////////////////////////////////////////
//   Implementation of Class MenuOpt              //
////////////////////////////////////////////////////
#include "./libuniv/Frame.h"
#include "MenuOpt.h"
#include "./libuniv/IconifyWin.h"
#include "./libuniv/QuitWin.h"
#include "University.h"
#include "Add.h"
#include "Print.h"
#include "Modify.h"
#include "Delete.h"
#include "Classes.h"
#include "Instances.h"
#include "Attributes.h"
#include "UserDefRel.h"
#include "GenericRel.h"
#include "Go.h"
#include "Cancel.h"
MenuOpt *theMenuOpt = new MenuOpt ( "MenuOpt" );
MainWin *univdb =
   new University ( "UNIVERSITY DATABASE BROWSER");

MenuOpt::MenuOpt ( char *name ) : Frame ( name )
{

// Create commands common to all windows
  _iconify = new IconifyWin ( "Iconify Window", TRUE);
  _quit    = new QuitWin ( "Quit Application", TRUE);

// Create Menu options for actions

_add     = new Add ( "Add", TRUE);
_print = new Print ( "Print", TRUE );
_modify  = new Modify("Edit", TRUE );
_delete  = new Delete( "Delete", TRUE );

// Create menu options for selection of type of entity
```

```
_classes = new Classes ( "Classes", TRUE );
_instances = new Instances ( "Instances", TRUE);
_attributes = new Attributes ( "Attributes", TRUE);
_userDefRel = new UserDefRel ( "UserDefRel", TRUE);
_genericRel = new GenericRel ( "GenericRel", TRUE);

// Create menu options to confirm action
_go = new Go ( "Go", TRUE );
_cancel = new Cancel ("Cancel", TRUE);
}

MenuOpt::~MenuOpt()
{
  //delete  _manage;
  delete  _iconify;
  delete  _quit;
  delete  _add;
  delete  _print;
  delete  _modify;
  delete  _delete;
  delete _classes;
  delete _instances;
  delete _attributes;
  delete _userDefRel;
  delete _genericRel;
 }

 ////////////////////////////////////////////////////
 //Class University                                //
 ////////////////////////////////////////////////////
 #ifndef UNIVERSITY_H
 #define UNIVERSITY_H
 #include "./libuniv/MenuWin.h"
 class ViewT1;
 class ViewT2;
 class ViewT3;
 class ViewT4;
 class ViewB1;
 class ViewB2;
 class Model;
 class Controller;
 class View;
```

```
class Iface;
class University : public MenuWin {

 protected:
 Widget _workArea;
 Widget _form, _acceptButton, _queryButton;
 Widget createWorkArea ( Widget );
 void createMenuPanes();
 static void acceptCallback(Widget, XtPointer,
XtPointer);
 static void queryCallback(Widget, XtPointer,
   XtPointer);
 public:
 University ( char * );

};
#endif

////////////////////////////////////////////////////
//   Implementation of Class University           //
////////////////////////////////////////////////////
#include "University.h"
#include "Controller.h"
#include "./libuniv/Iface.h"
#include "View.h"
#include "ViewT1.h"
#include "ViewT2.h"
#include "ViewT3.h"
#include "ViewT4.h"
#include "ViewB1.h"
#include "ViewB2.h"
#include "Model.h"
#include "./libuniv/MenuBar.h"
#include "MenuOpt.h"
#include "./libuniv/Commands.h"
#include <Xm/Xm.h>
#include <Xm/Form.h>
#include <Xm/RowColumn.h>
#include <Xm/CascadeB.h>
#include <Xm/PushB.h>
//#include "./libuniv/MenuBar.h"
#include <iostream.h>
```

```
University::University ( char *name ) :
   MenuWin ( name )
{
}

Widget University::createWorkArea ( Widget parent )
{
Arg Args[5];
int i;
ViewT1 *_viewT1;
ViewT2  *_viewT2;
ViewT3  *_viewT3;
ViewT4  *_viewT4;
ViewB1  *_viewB1;
ViewB2  *_viewB2;
Model   *_model;
Controller *_controller;

_workArea = XtCreateWidget ( "form",
                             xmFormWidgetClass,
                              parent,
                             NULL, 0 );
installDestroyHandler();
XtVaSetValues(_workArea, XmNwidth, 800,
                          XmNheight, 600,
                          NULL);
_acceptButton  = XtVaCreateManagedWidget("accbut",
                          xmPushButtonWidgetClass,
                          _workArea,
                          XmNlabelString,
                          XmStringCreateSimple("Accept"),
                          NULL);

_queryButton  = XtVaCreateManagedWidget("querybut",
                         xmPushButtonWidgetClass,
                         _workArea,
                         XmNlabelString,
                         XmStringCreateSimple("Query"),
                         NULL);

_viewT1 = new ViewT1("viewT1", _workArea);
_viewT2 = new ViewT2("viewT2", _workArea);
```

```
_viewT3 = new ViewT3("viewT3", _workArea);
_viewT4 = new ViewT4("viewT4", _workArea);
_viewB1 = new ViewB1("viewB1", _workArea);
_viewB2 = new ViewB2("viewB2", _workArea);

XtAddCallback(_acceptButton, XmNactivateCallback,
        &University::acceptCallback, (XtPointer) _viewT4);

XtAddCallback(_queryButton, XmNactivateCallback,
        &University::queryCallback, (XtPointer) _viewB1);

XtVaSetValues( _viewT1->rootWidget(), XmNtopAttachment,
    XmATTACH_FORM, XmNleftAttachment,
            XmNleftAttachment, XmATTACH_FORM,
            XmNrightAttachment, XmATTACH_POSITION,
            XmNrightPosition, 25,
            XmNbottomAttachment, XmATTACH_POSITION,
            XmNbottomPosition, 40, NULL);

XtVaSetValues( _viewT2->rootWidget(), XmNleftAttachment,
 XmATTACH_POSITION,
                XmNleftPosition, 25,
                XmNrightAttachment, XmATTACH_POSITION,
                XmNrightPosition, 50,
                XmNtopAttachment,  XmATTACH_FORM,
                XmNbottomAttachment, XmATTACH_POSITION,
                XmNbottomPosition, 40, NULL);
XtVaSetValues( _viewT3->rootWidget(), XmNtopAttachment,
     XmATTACH_FORM,
                XmNbottomAttachment, XmATTACH_POSITION,
                XmNbottomPosition, 40,
                XmNrightAttachment, XmATTACH_POSITION,
                XmNrightPosition, 75,
                XmNleftAttachment, XmATTACH_POSITION,
                XmNleftPosition,   50, NULL);
XtVaSetValues( _viewT4->rootWidget(), XmNtopAttachment,
     XmATTACH_FORM,
                XmNbottomAttachment, XmATTACH_POSITION,
                XmNbottomPosition, 35,
                XmNrightAttachment, XmATTACH_FORM,
                XmNleftAttachment, XmATTACH_POSITION,
                XmNleftPosition,   75, NULL);
```

```
XtVaSetValues( _acceptButton, XmNtopAttachment, XmATTACH_POSITION,
                  XmNtopPosition, 35,
                  XmNbottomAttachment, XmATTACH_POSITION,
                  XmNbottomPosition, 40,
                  XmNleftAttachment, XmATTACH_POSITION,
                  XmNleftPosition, 75,
                  XmNrightAttachment, XmATTACH_FORM, NULL);
XtVaSetValues ( _viewB1->rootWidget(), XmNleftAttachment,
     XmATTACH_FORM,
                  XmNbottomAttachment, XmATTACH_POSITION,
                  XmNbottomPosition, 95,
                  XmNtopAttachment, XmATTACH_POSITION,
                  XmNtopPosition, 40,
                  XmNrightAttachment, XmATTACH_POSITION,
                  XmNrightPosition, 50, NULL);

XtVaSetValues( _queryButton, XmNtopAttachment, XmATTACH_POSITION,
                  XmNtopPosition, 95,
                  XmNbottomAttachment, XmATTACH_FORM,
                  XmNleftAttachment, XmATTACH_FORM,
                  XmNrightAttachment, XmATTACH_POSITION,
                  XmNrightPosition, 50,
                  NULL);
XtVaSetValues ( _viewB2->rootWidget(), XmNtopAttachment,
     XmATTACH_FORM, XmNtopPosition, 40,
                  XmNbottomAttachment, XmATTACH_FORM,
                  XmNrightAttachment, XmATTACH_FORM,
                  XmNleftAttachment, XmATTACH_POSITION,
                  XmNleftPosition, 50, NULL);

_model = new Model( _viewT1, _viewT2, _viewT3, _viewT4, _viewB1,
 _viewB2 );
_controller = new Controller( _model );
return (_workArea);

}
////////////////////////////////////////////////////////////
// Callback function for Accept Button of Edit Window
////////////////////////////////////////////////////////////
void University::acceptCallback(Widget, XtPointer clientData,
XtPointer)
{
```

```
 String _inText = NULL;
 ViewT4 *obj = (ViewT4 *) clientData;
  //_inText = strdup(XmTextGetString(obj->rootWidget());
// This function does not seem to
 // theController->registerFromViewT4(_inText);
 // exist in the system
}

////////////////////////////////////////////////////////////
// Callback function for Query Button of of Window-5
////////////////////////////////////////////////////////////
void University::queryCallback(Widget, XtPointer clientData,
   XtPointer)
{
 String _inText = NULL;
 ViewB1 *obj = (ViewB1 *) clientData;
  //_inTexts = XmTextGetString(obj->rootWidget());
  //This function does not seem to
 //theController->registerFromViewB1(_inText);
 //exist in the system
}

////////////////////////////////////////////////////////////
// Creating the menu panes
////////////////////////////////////////////////////////////
void University::createMenuPanes()
{
 Commands *commands;

 commands = new Commands();
 commands->add (theMenuOpt->iconifyCmd() );
 commands->add (theMenuOpt->quitCmd() );
 _menuBar->addCommands (commands, " SYSTEM " );

 delete commands;


 // Create command list consisting of add, print, modify and
 // delete and a menu pane with the name "OPTIONS" to contain
 // these commands

 commands = new Commands();
```

```
 commands->add ( theMenuOpt->addCmd() );
 commands->add ( theMenuOpt->printCmd() );
 commands->add (theMenuOpt->modifyCmd() );
 commands->add (theMenuOpt->deleteCmd() );
 _menuBar->addCommands ( commands, " ACTION ");
 delete commands;

 // Create Command list for diffrent selections
 // (instances, subTypeOf, roleOf, CategoryOf, attributes,
 //   relationships)

 commands = new Commands();
 commands->add ( theMenuOpt->classesCmd() );
 commands->add ( theMenuOpt->instancesCmd() );
 commands->add ( theMenuOpt->attributesCmd() );
 commands->add ( theMenuOpt->userDefRelCmd() );
 commands->add ( theMenuOpt->genericRelCmd() );
 _menuBar->addCommands ( commands, "SELECTION");
 delete commands;
 // Create Command list for confirming the action ( go, cancel)
 commands = new Commands();
 commands->add ( theMenuOpt->goCmd() );
 commands->add ( theMenuOpt->cancelCmd() );
 _menuBar->addCommands ( commands, "CONFIRM ");
 delete commands;
 // To be continued for other sets of commands //


}

//////////////////////////////////////////////////
// Class ViewB1                                 //
//////////////////////////////////////////////////
#ifndef VIEWB1_H
#define VIEWB1_H
#include "View.h"
class ViewB1 : public View {

 private:
        Widget _scrollTextB1;
        char _inText[1000];
```

```
 public:
       ViewB1 ( char *name, Widget parent);
       void update ( Model *model);
       virtual const char *const className()
   { return ( "viewB1" ); }
};
#endif

//////////////////////////////////////////////////
//   Implementation of Class ViewB1             //
//////////////////////////////////////////////////
#include "View.h"
#include "ViewB1.h"
#include "Controller.h"
#include "Model.h"
#include <Xm/Xm.h>
#include <Xm/Text.h>
#include <iostream.h>
ViewB1::ViewB1(char *name,  Widget parent) : View( name )
{
  _scrollTextB1 = XmCreateScrolledText( parent,
  "scrollTextB1",
NULL,
   0);

_w = XtParent(_scrollTextB1);
installDestroyHandler();
XtVaSetValues ( _w, XmNscrollBarPlacement, XmBOTTOM_LEFT,
                  NULL);
XtVaSetValues(_scrollTextB1, XmNeditable, True,
                   XmNeditMode, XmMULTI_LINE_EDIT,
                   XmNmaxLength, 2000,
                    NULL);


XtManageChild(_scrollTextB1);
XtManageChild(_w);
}

//////////////////////////////////////////////////////////
// This function is invoked my Model class for updating the
// view buffer
```

```
////////////////////////////////////////////////////////////

void ViewB1::update( Model *model ) {
 int i;

 XtVaSetValues(_scrollTextB1,
              XmNvalue, model->_outText,
               NULL );
// Access data from model
}
```

REFERENCES

[B90] A. Bhave, "Implementation of University Database using the VML – The Object–Oriented Database System (Release 1)", *Master's Thesis, CIS–Dept*, NJIT, Newark, NJ, 1990.

[BOS91] Butterworth, P., Otis, A., and Stein, J., "The Gemstone Object Database Management System", *Communications of the ACM, vol. 20*, Oct. 1991, pp. 64–77.

[C92] Chatterjee, S., "Graphical Image Persistent and Code Generation for Dual Model Object Oriented Databases", *Master's Thesis*, CIS Department, NJIT, Newaark, NJ, 1992.

[CT90] Chao, H., Teli, V., "Development of a University Database using the Dual Model of Object–Oriented Knowledge Bases", *Master's Thesis*, CIS Department, NJIT, Newark, NJ, 1990.

[D91] Dixit, N., "Implementation of University Database using the VML – The Object-oriented Database System (Release 2)", *Master's Project*, CIS–Dept, NJIT, Newark, NJ, 1991.

[GPN91] Geller, J., Perl, Y., and Neuhold, E.J., "Structure and Semantics in OODB class Specifications", *SIGMOD Record, Special Issue on 'Semantic Issues in Multidatabase Systems', vol. 34*, Dec. 1991, pp. 40–43.

[GR83] A. Goldberg, D. Robson, "Smalltalk-80, The Language and Its Implementation", Addison Wesley, 1983.

[HGPN92] Halper, M., Geller J., Perl, Y., Neuhold, E. J., "A Graphical Schema Representation for Object–Oriented Databases", *In the Proceedings of*

IDS92, International Workshop on Interfaces to Database Systems, Glasgow, July 1992.

[K90] S. Kulkarni, "Implementation of University Database using the VML – The Object–Oriented Database System",*Master's Project*, CIS–Dept, NJIT, Newark, NJ, 1990.

[KNBD92] Klas W., Neuhold E. J., Bahlke, R., Drosten K., Fankhauser P., Kaul M., Muth P., Oheimer M., Rakow T., Turau V., "VML Design Specification Document", *Tech Report, GMD–IPSI*, Germany, 1992.

[M91] Martin, R., "ONTOS Overview", *In the Proceedings of Executive Briefing on Object–Oriented Database Management*, San Fransisco, 1991.

[NPGT89] Neuhold, E.J., Perl, Y., Geller, J., Turau, V., "Separating Structural and Semantic Elements in Object–Oriented Knowledge Bases", *Advanced Database System Symposium*, Kyoto, Japan, 1989, pp. 67–74.

[M93] Mehta, A., "Algorithms for Generation of Path–Methods in Object–Oriented Databases", *Doctoral Dissertation*, Computer Science Department, New Jersey Institute of Technology, May 1993.

[NPGT90] Neuhold, E.J., Perl, Y., Geller, J., Turau, V., "A Theoretical Underlying Dual Model for Knowledge Based Systems", *In the Proceedings of the First International Conference on Systems Integration*, Morristown, NJ, 1990, pp. 96–103.

[NPGT91] Neuhold, E.J., Perl, Y., Geller, J., Turau, V., "The Dual Model for Object–Oriented Databases", *Research Report 91–30*, CIS Department, NJIT, 1991, Submitted for Publication.

[OHMS92] Orenstein, J., Haradhwala, S., Margulies, B., Sakahara, D., "Query Processing in the ObjectStore Database System", *In the Proceedings of*

the 1992 ACM SIGMOD International Conference on Management of Data, San Diego, California, June 2–5, 1992, pp. 403–412.

[O91] The Open Software Foundation "OSF/Motif Programmer's reference", *Open Software Foundation, Cambridge, MA*, 1990.

[Y92] Young, D. "Object Oriented Programming with C++ and OSF/Motif", *Prentice Hall*, 1992.

[P91a] Pandit, H.,"Implementation of University Database using the C++ Programming Language", *Master's Project*, CIS–Dept, NJIT, Newark, NJ, 1991.

[P91b] Patel, M., "Implementation of University Database using the VML – The Object–Oriented Database System (Release–3)", *Master's Project*, CIS–Dept, NJIT, Newark, NJ, 1991.

[S91] B. Stroustrup, "The C++ Programming Language", *Second Edition, Addison Wesley Publishing Company,Reading MA.*, 1991.