BROWN UNIVERSITY
Department of Computer Science
Master's Project
CS-89-M11

"Graphical User Interface to ENCORE --Type and Instance Browser"

by
Young Woo Koh

Graphical User Interface to ENCORE

— Type and Instance Browser

Young Woo Koh

Department of Computer Science
Brown University

Submitted in partial fulfillment of requirements
for the degree of Master of Science in the
Department of Computer Science at Brown University

December, 1989

This project by Young Woo Koh
is accepted in its present form by the Department of
Computer Science in partial fulfillment of the
requirements for the degree of Master of Science.

Date 12/27/89

Stanley B. Zdonik

Professor Stanley B. Zdonik
Advisor

Contents

Graphical User Interface to ENCORE

Young Woo Koh

December 21, 1989

Abstract

This paper presents a graphical interface to **ENCORE**, an Object-Oriented Database system being developed at Brown University. The overall design criteria and a description of interface will be discussed. In particular, *Schema* and *Instance* browsing and modification will be described in detail. The *Schema* browser allows a user to browse through the *type* system of ENCORE by following chains. Similarly, the *Instance* browser allows the navigation of instances, which are the actual data. Our browser utilizes the location of interesting data through a series of selections on a large number of objects, thus, providing an easy and convenient way to narrow the range of data of interest.

1 Introduction

Advances in window systems, availability of high resolution bitmapped displays, and pointing devices such as the mouse make it easy to build sophisticated graphical displays of data. The primary purpose of graphically displaying data is to provide an easy and convenient way to access and manipulate data. Often people understand better the logical structure of data through visualization than through any other representation. Another advantage of a graphical interface is that it gives immediate feedback of the user's actions on data. For example, when the user adds a new instance, the change is immediately reflected graphically so that the user can see that the instance has been added. Many traditional database interfaces, usually textual, fail to provide

an easy use of the underlying system. Browsing data through a sequence of query operations is certainly not convenient and it is especially so for novice and infrequent users. The forming a query in textual interface is not only difficult, but also leads to errors such as simple typing mistakes, which are not detected until execution time. The graphical interface aims to provide an easy and convenient way to use a database system without requiring the user to have extensive knowledge about the underlying system or query language.

This document describes the graphical user interface(**GUI**), to the ENCORE[15] object oriented database system. Using GUI, the user can graphically browse the type structure and instances of ENCORE. It also provides an interface to operation invocation and some of the schema design facility. More formally, GUI consists of 3 main components:

1. The **Schema interface** is divided into two parts:
   - **Schema design** allows the user to create new types and modify existing ones.
   - **Schema browser** allows the user to browse through the existing types of ENCORE.
2. The **Instance interface** also has two parts:
   - **Instance design** allows the user to create new instances and modify existing ones.
   - **Instance browser** allows the user to browse through the instances of ENCORE.
3. The **Query Interface** allows the user to operate on the data via queries made in the operation invocation facility. Unfortunately, at this time, most of query interface is textual rather than graphical.

This report mainly concerns Type and Instance Browser, which the author designed and implemented. A more detailed report on the Query and Type Hierarchy can be found in [5]. Functionalities of GUI are discussed in more detail in Section 2.

1.1 Related Work

There has been some work on graphical user interfaces to database systems in the last 10 years. **CUPID, QBE**[16], and **GUIDE**[8] are graphical query languages to relational databases. In QBE, the user forms a query by filling in a skeleton of table in QBE. In CUPID, the user constructs a query by graphically using graphical elements such as rectangles, circles which represent the query components and comparison operators. Even though both systems provide some form of graphical aid in query operation, they do not provide a facility for browsing the schema. Also, the user cannot browse the answers of query operations in those systems. In GUIDE, the database schema is displayed as a network of entity relation types. The user expresses a query as a traversal path on the network. The GUIDE does provide a facility of browsing meta-data, the E-R schema. Browsing Meta-data gives general view of the data, instances, even though the user has no direct access to instances. That is, the browsing is restricted to only schema level. It is hard to directly compare GUI with the systems mentioned above. Since they are graphical query languages, they provide similar features of Query Interface of GUI. However, browsing results of a query operation is not incorporated in these systems while the Query Interface in GUI is closely related to both Schema and Instance browser.

SDMS[6] provides more sophisticated browser of actual data. SDMS provides the graphical view of actual data so that user can directly manipulate data through graphical images. SDMS shows the entire database in several screens, from which the user can browse data using a joystick. Unlike other systems mentioned, it presents data at several different levels. For example, one screen shows entire database to give an overall picture and another screen designated for a detailed view allows the user to examine details of the data. detailed view. This systems, however, lack a schema browser, which is necessary to guide the user through complex database. Also, the comparison between data is not convenient unless they were displayed together to begin with. That is, the user has no control over the layout of data. More recently, **PICASSO**[7] is a graphics-based database query language derived from the universal relation database system. The user can form quite complex queries and immediate feedback is available during the process of forming query. However, as with other systems mentioned, the result of query is textual and the user cannot browse the result.

ISIS[4] and **SNAP**[1] are graphical interfaces to a semantic data model. These systems allow graphical schema design, browsing, and query operation. In each system, starting from a subset of a global hierarchy of schema, the user can interactively examine the detail of data graphically. The user of ISIS can browse results of a query at the data level while the results of a query in SNAP are presented textually and can not be browsed. In ISIS, the browsing of schema and data are combined together where schema and data are displayed side by side, while SNAP only provides schema browsing. An interesting feature of schema browser of SNAP is that the user can reposition, hide, and redisplay objects. Both systems introduce useful features, particularly in the query interface which GUI cannot provide at this point. However, ISIS shows an inconsistency where browsing the schema is treated differently from browsing at the data level. As mentioned above, SNAP lacks a connection between query interface and instance interface. Also, neither systems provides a global hierarchy facility as in GUI though they show a subset.

There has been some documented research on object-oriented database systems. **SIG**[11] has type model similar to that of Smalltalk. SIG allows the user to customize the way objects are displayed. One interesting feature of SIG is that it provides an update facility on multiple views of same database object. This update facility was not supported by previous systems. Another feature is that it allows the customization of the layout of objects. Since GUI currently does not have these features, their addition would certainly enhance GUI. However, SIG lacks schema browsing and a query facility which is necessary in complex database. Its main concern is automatic generation of graphical display of data from descriptions.

In general, each system introduces a useful feature, but none has all the features identified as important for a good graphical database interface.

1.2 Data Model

The data model for **ENCORE** is based on the notion of a type. A **Type** is a behavioral template for instances of that type. In **ENCORE**, a type definition is composed of three components: **P**, a set of properties, **O**, a set of operation, and **C**, a set of constraints. A **Property** is an object that is used to relate types to other types. An **Operation** is a procedure that can be invoked

through method invocation and is the user's only interface to instances of the type definition. Operations insure the encapsulation of objects. Finally, the **constraints** are a set of conditions that has to be maintained for the type during its lifetime.

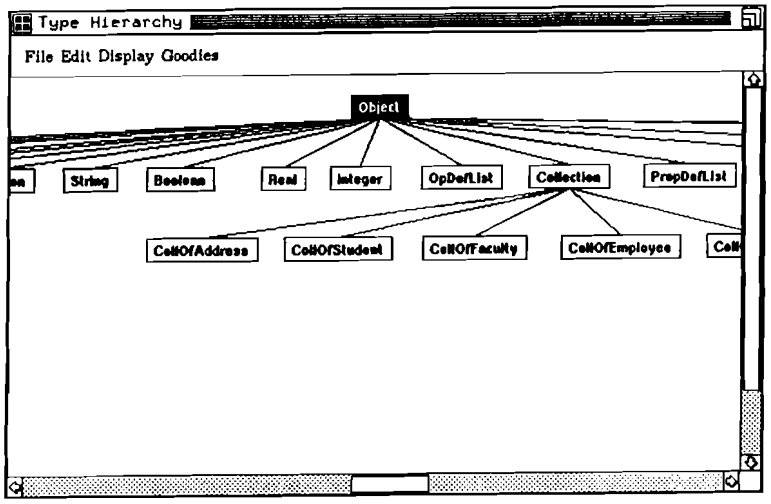
In **GUI**, the type definition is represented as a quadruple , **type** = (**T**, **S**, **P**, **O**). The constraints are not represented as separate items, but are understood within the context of a given type definition. **T** is the name of the type, **S** is a list of supertypes for **T**, **P** is a list of properties and finally, and **O** is a list of operations defined on that type. Since ENCORE supports multiple inheritance, a type can have more than one supertype. In displaying a type, we only include the immediate parents as supertypes. However, both property and operation sections display all the properties and operations including those inherited. Allowing multiple inheritance makes implementation more difficult, but is a more elegant way for software reuse than single inheritance. As will be described late, our **Type Hierarchy** shows **is-a** relationships between types including multiple inheritance.

2 GUI Functionality

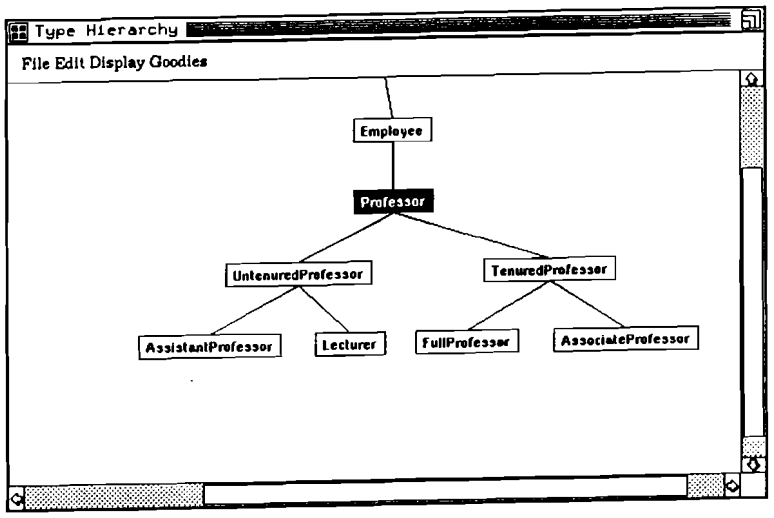
This Section describes briefly the functionalities of **GUI**. Currently, 5 different windows are used, one for each function: **Type Hierarchy, Type Tracking, Type Static, Instance Static** and **Operation Invocation**. Following sections will discuss each of the 5 windows, particularly, the Type Static and Instance Static windows.

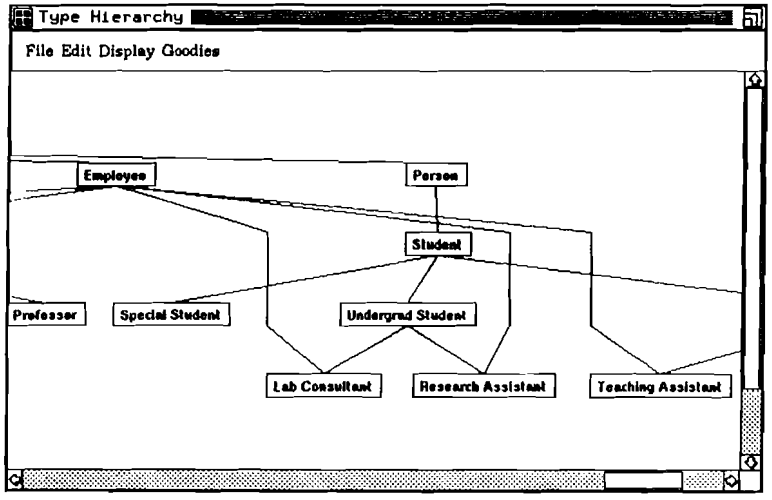
2.1 Type Hierarchy and Type Tracking Window

As in other object-oriented database systems ENCORE's type structure is very complicated. Grasping the relationships among types without any help is very difficult because the user does not know where and how to start. It is necessary to allow the user to explore the schema of the underlying system so that the user understands the overall structure of data in the system. The user can graphically browse the entire schema of ENCORE in the **Type Hierarchy** window. Every type in ENCORE is connected by an is-a, supertype-subtype

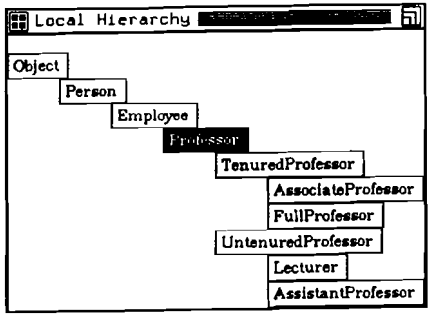


(a) Global hierarachy graph





(b) Multiple Inheritance display



(c) Local hierarchy

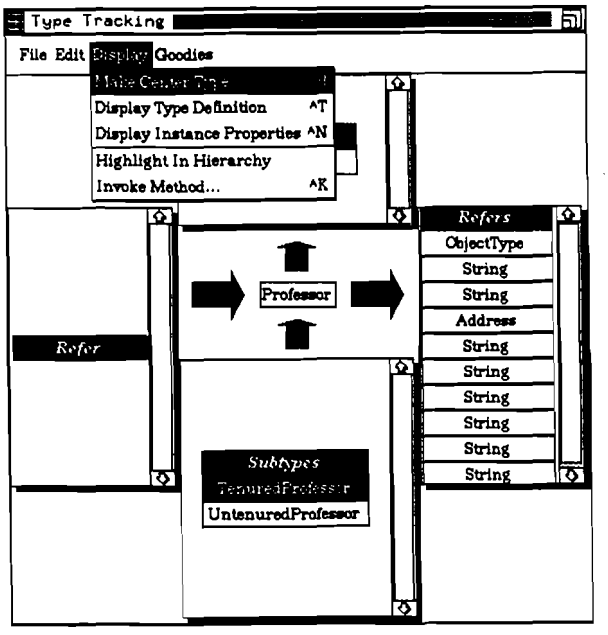
Figure 1. Type Hierarchy Window

relationship. Type **Object** is the highest object and it is supertype of all other types in the system. Figure 1(a) and (b) show two different portions of an example type system. In these figures, the boxes indicate types and the line shows the is-a relationships. A picture of **TypeHierarchyWindow** with an example type system is shown in Figure 1(a). In Figure 1(b) the reader should notice that *Lab Consultant* has two supertypes, *Employee* and *Undergrad student*, which shows multiple inheritance. If there is more than one supertype for a type, then that type has edges from each of those supertypes. Since the window can not show entire the schema at one time, the user examine the schema using both horizontal and vertical scrollbar attached to the window. Another way to focus on objects of interest is to have localized view of a hierarchy centered on a particular type. The Figure 1(c) shows the localized hierarchy of type *Professor*. Each type can be selected, and detailed information about the type can be examined in the Type Tracking or Type Static window.

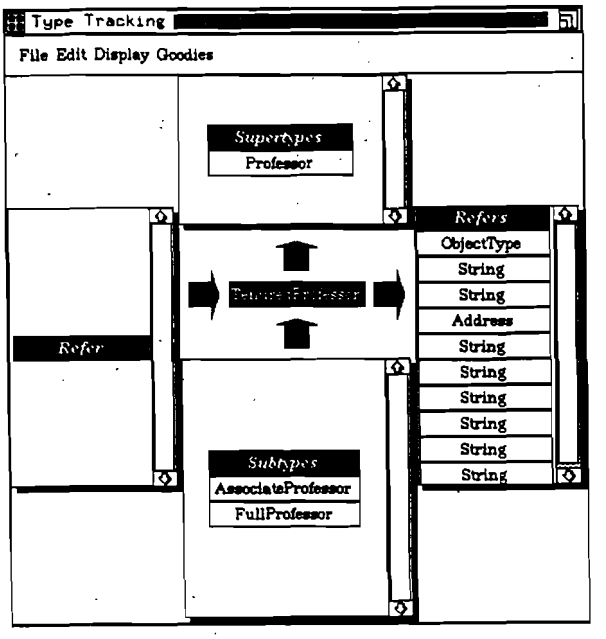
The Type tracking window shows a localized view of the relationship created between a type and another type as its property value. This window is opened from Type Hierarchy window when the user selects a type. The selected type is displayed in the center of this window along with 4 other relationships:

1. A list of supertypes is displayed at the top of the selected type.
2. A list of subtypes of the selected type is displayed at the bottom.
3. A list of types that have the selected type as a property value are displayed on the left side.
4. Types that the given type has as property values are displayed on the right side.

The **TypeTrackingWindow** is shown in Figure 2(a), where *Professor* is the center type and the 4 relationships are displayed accordingly. When the user selects *TenuredProfessor* to make it the center type, the 4 relationships are changed accordingly as in Figure 2(b). Notice that *Professor* is the supertype of *TenuredProfessor*. This facility is very useful when the user wants to examine all relationships of a particular type, which are not shown in the global



(a) Professor as center type



(b) TenuaredProfessor as Center type

Figure 2. Type Tracking Window

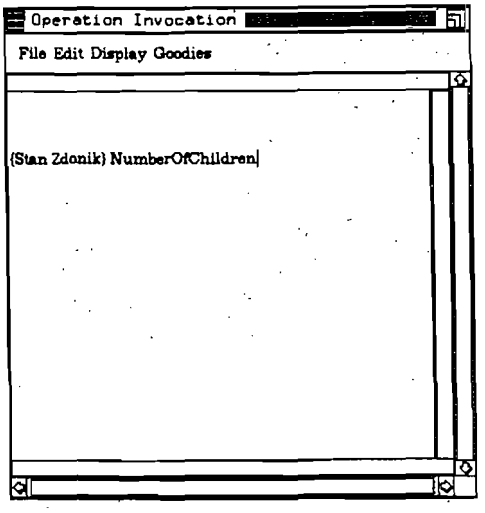


Figure 3. Operation Invocation Window

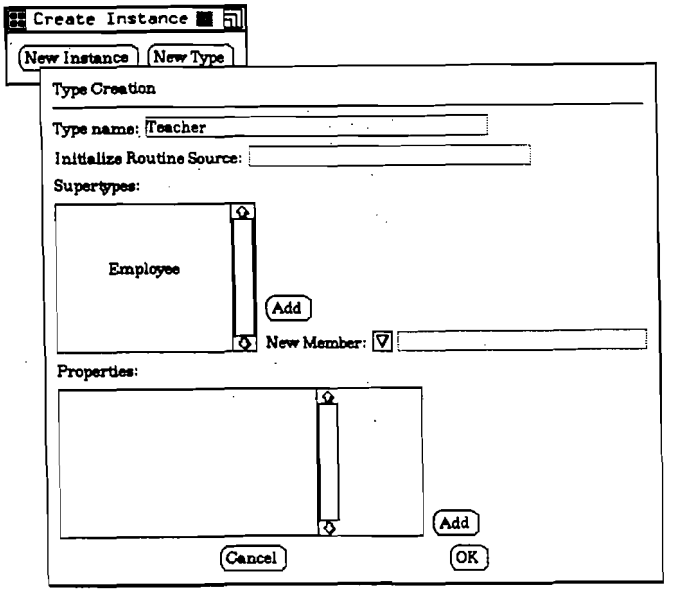


Figure 4. Type Creation Dialog Box

hierarchy. The user can change the center object by issuing the **Make Center Type** on a selected type within the window. Four relations with a new center object are updated accordingly.

2.2 Operation Invocation

Operation invocation in GUI is done textually. Query operations are done in the **Operation Invocation** window. Operation invocation window is shown in Figure 3. The user formulates an operation invocation string by specifying an object, a name of an operation, and parameters. The object is represented by a string enclosed by curly braces. For example, the user wants to invoke the operation *NumberOfChildren* on the object, *Stan Zdonik* in Figure 3. Forming the invocation string is done by **copy** and **paste** functions using the clipboard window. Since an invocation string can be long and complicated for a fairly complex operation, it would be appropriate to formulate the string incrementally. That is, the user can form a subsequent string based on the previous intermediate results. The result of each operation is displayed in the Operation Invocation window and the result can be examined graphically in the **Instance Static** window. In this way, operation invocation can be effectively combined with a browser.

2.3 Type and Instance Creation

In ENCORE, creating a type is equivalent to creating an instance of type, **Type** because every type is an instance of type, **Type**. To create a type, the user specifies names of supertypes so that the properties and operations of the supertypes are inherited to the type to be created. Its own properties and operations are added by explicitly defining them in the Type creation dialog box. The type creation dialog box is shown in Figure 4, where the user is creating a new type *Teacher* and just added *Employee* as its supertype. Currently, a facility for defining operations for type creation is not provided, but should be added in near future. Instances are created in a similar dialog box when the user select a type and issues **Create Instance**. Assigning values for the properties of the selected type is achieved either by explicitly typing in the value or by selecting objects from list of instances of the appropriate type.

If a value is of primitive type, the user can simply type the value. Otherwise, the user must select an object for a value from existing list of instances if one exists. If it does not exist, a new instance must be created and the level of nesting can go on until there is no need for creation of instances as values.

2.4 Type Browser Window

Once the user gets familiar with the overall type structure, the next step is to examine the contents of an individual type. The user usually selects a type from the **Type Hierarchy** or **Type Tracking** window and opens a **Type Static** window by issuing **ShowType** to examine its contents. The full view of the selected type is displayed at the left in the window and each type object is represented as a rectangular box with subdivision as in Figure 5(a). The top of the box shows the name of type, a property of type **Type** in ENCORE. Below the name, it shows the supertypes of the type. Properties of the type are shown below and a list of operations for the type is shown in last portion of the box. In Figure 5(a), *TenuredProfessor* is the name of the type and *Professor* is its supertype. Since window space is limited while the number of properties or operations for a type can be large, we initially display a portion of entire definition of the type, and the rest of the definition can be examined by using the scrollbars attached to each part of the type object. Each item in the supertype list is the name of a supertype. Each property item shows the name of a property and an icon. There are three different types of icons, which indicate the kinds of types in ENCORE. Primitive types like Integer, Real are represented by a circle with a character, "**s**" inside it, complex types by a triangle, and collection types by a double triangle. Figure 5(a) shows that the *address* property is of complex type while *firstName* is of primitive type. Each operation item also has a corresponding icon for its return type. The name and return type of an operation are both displayed.

To examine contents of a supertype or a property, the user selects the desired item and issues **Follow Chain**. The selected item becomes highlighted and a full view of the selected item is displayed next to the object containing the item. Also, an edge is drawn to the newly displayed object from the object containing the selected item to indicate a derivation. Figure 5(c) shows a chain of type objects, starting from *TenuredProfessor* to *Person*. When the

user issues **FollowChain** after selecting **Professor**, it displays the *Professor* type definition next to it. When an additional full view of a type is chained from the same type object, the existing display is collapsed to a simple object. If the collapsed object has chains following it, an arrow will be displayed in that simple object so that user can recognize that there are further chains as in Figure 5(d), where *Employee* type is collapsed and it has an object, *Person* following. The simple object can be expanded to a full view when the **Expand** command is executed. By default, only one full view is displayed for all the types referenced from one type object. However, there is a command **ExpandAll** which will expand a selected simple object to its full view while the current full view object remains the same. The **ExpandAll** command is useful when the user wants to compare two type objects. Figure 5(b) shows that *Address* and *String* types are displayed in their full view. There are two commands for the operation object, **Show Parameter** and **Show Definition**. **Show Parameter** displays an object with list of parameters for an operation and each parameter can be examined by following the chain as with supertypes and properties. Source code for an operation is displayed in Source window when **Show Definition** is issued.

The number of objects in a chain is not limited. The user may also display more than one chain of type objects by executing **NewChain**. A full view of the selected type is displayed below the last chain and it becomes the head of a new chain.

Any object in the Type Static window can be removed by executing **RemoveObject**. This does not delete the selected object from the database, but removes the corresponding display object. If the deleted object has objects following, all objects along its chain are also removed. The user may also remove objects from the display with the **Simplify** which will explicitly shrink a full view object to simplified object with only the name of type displayed. If there is a further chain following that type, all the following objects are removed from the display. An arrow is inserted into the simplified object to indicate that there are further chains. However, this differs from **RemoveObject** since those objects are not actually deleted from the display list, but temporarily invisible. Therefore, all these invisible objects can be redisplayed when simplified object is expanded to its full view. This feature is useful because the user may want to save space, but not want to delete whole chains of an object. This is specially true if the deleted chain is very long because rebuilding the

chain will take time.

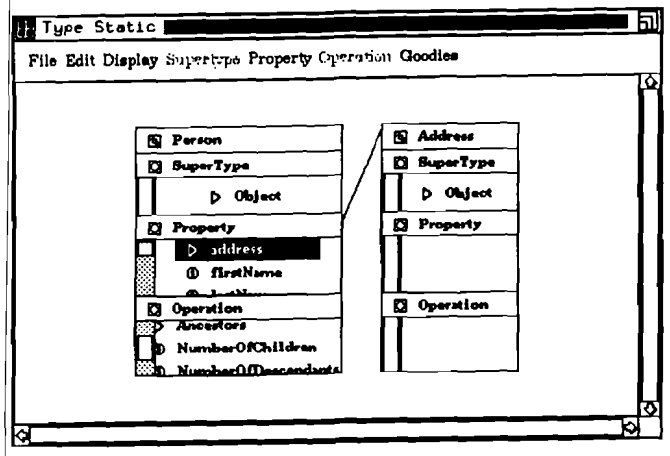
Since inheritance is a very important mechanism in an object-oriented system, visualizing the inheritance relationship between type objects is a necessary feature. The **ShowProp** and **ShowOper** will show properties and operations inherited from the selected supertype by dimming out all others. **ShowAll** display both properties and operations for a supertype.

When the user issues **ShowInst** on a type object, a new window is opened to show the list of instances for the selected type. This window is **Instance Static** window and individual instances can be explored in the window.

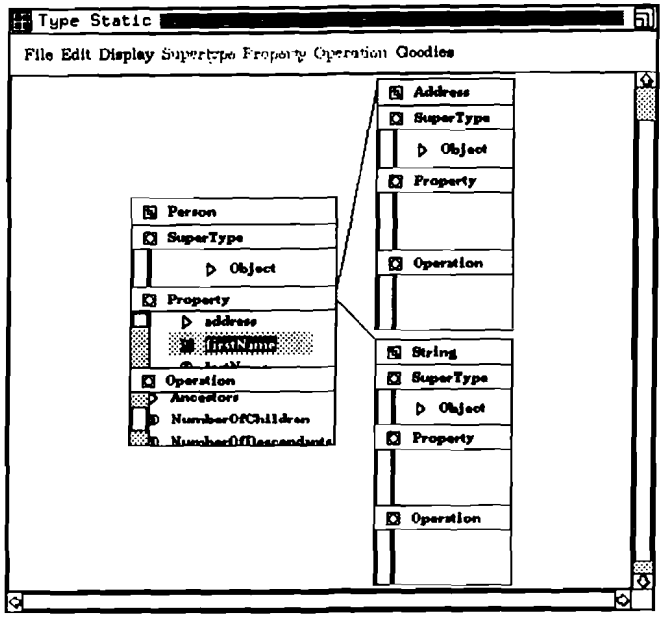
2.5 Instance Browser Window

As with type objects, each instance is represented by a rectangle and the name of the instance is shown at the top. Property values for each of the instances are displayed below the name. Figure 6(a) shows the Instance browser, where the *Department* instance is shown as the head of a chain. Primitive types such as Integer, Real, and Boolean are displayed as actual values (see Figure 6(a), where the value is a telephone number of *Stan Zdonik*). For a value of a Complex type, which is indicated by single triangle, the objects' **display name** is shown and the user can examine it further by following chains. Each type in ENCORE defines a display name property which will contain a string to be displayed for an instance of Complex or Collection type. The example shows that *StanZdonik* is of complex type while *CSfaculty* is of collection type. *StanZdonik* is a value of *CSfaculty*. The display name is not used as an identifier for an object, but as descriptive name for an object. As in the Type Browser, a new edge is drawn to the newly displayed instance. Existing instances with a full view are collapsed and following instances are removed as in the Type Browser. A property value of **ColType** is indicated by double triangle and when it is expanded, it displays an object containing the list of instances which are the values for that property.

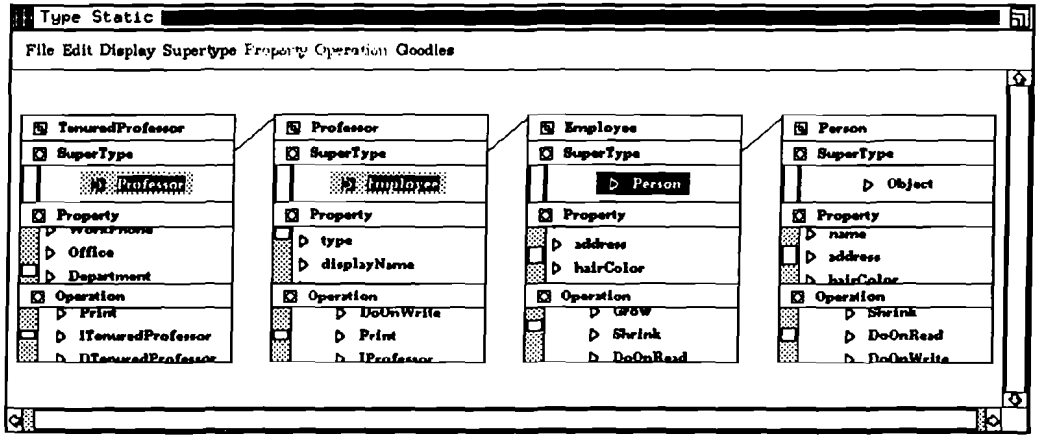
Edges along a chain connect those instances which belong to the chain. There can be,however, ambiguity concerning which instances refer to which other instances. When an instance is selected, it is fully highlighted while instances selected along the chain are halftone-highlighted as long as it remains in the browser. Thus, following the halftoned items, the user can easily rec-



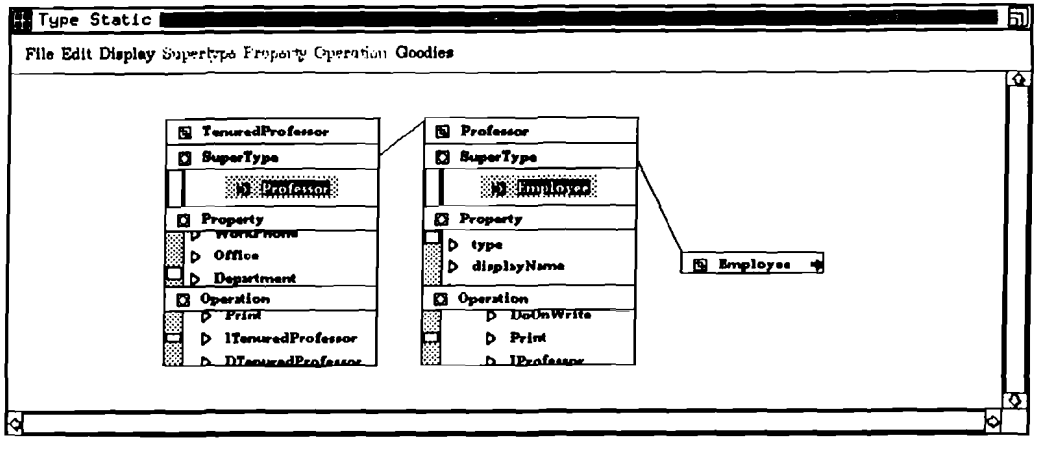
(a) After address is followed



(b) Both String and Address types are displayed in full view

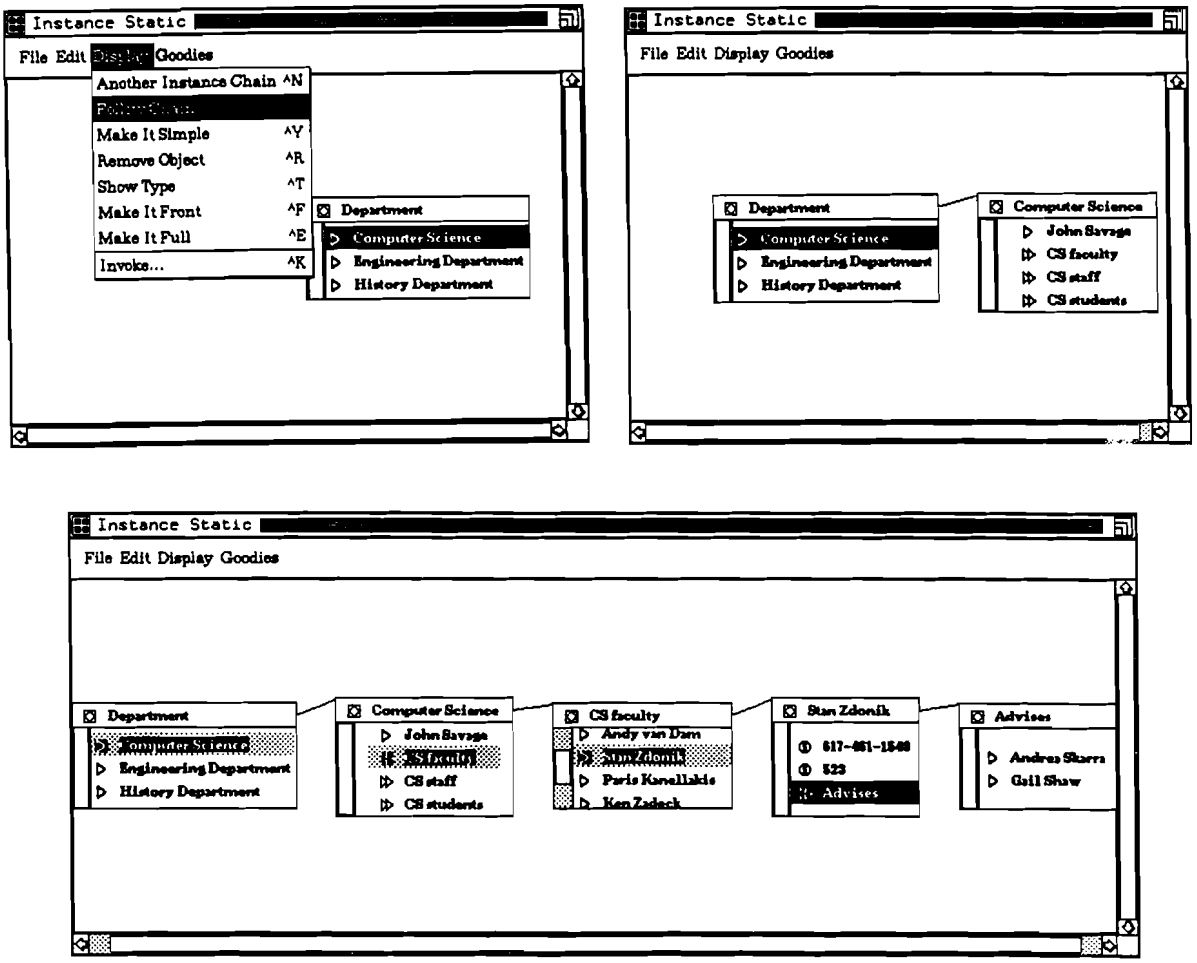


(c) A chain from TenuaredProfessor

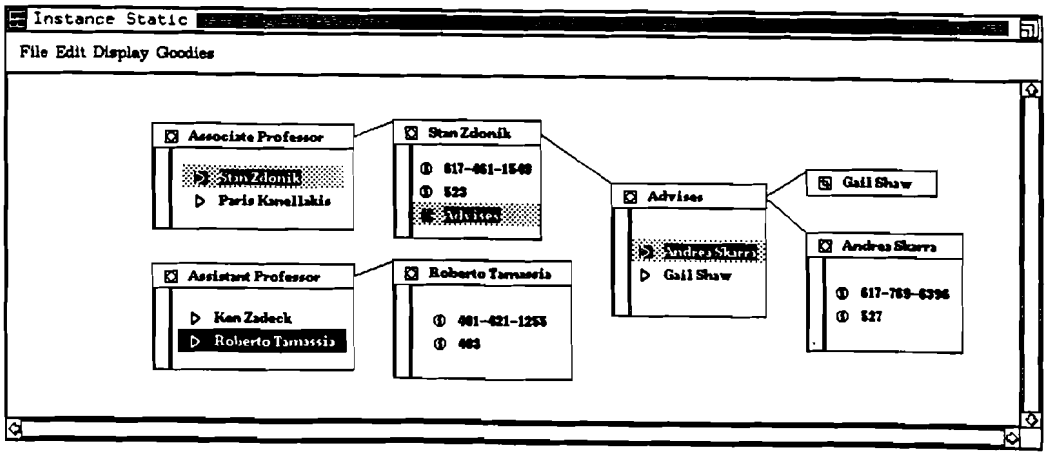


(d) After simplification of Employee type

Figure 5. Type Browser



(a) Instance chain from Department object



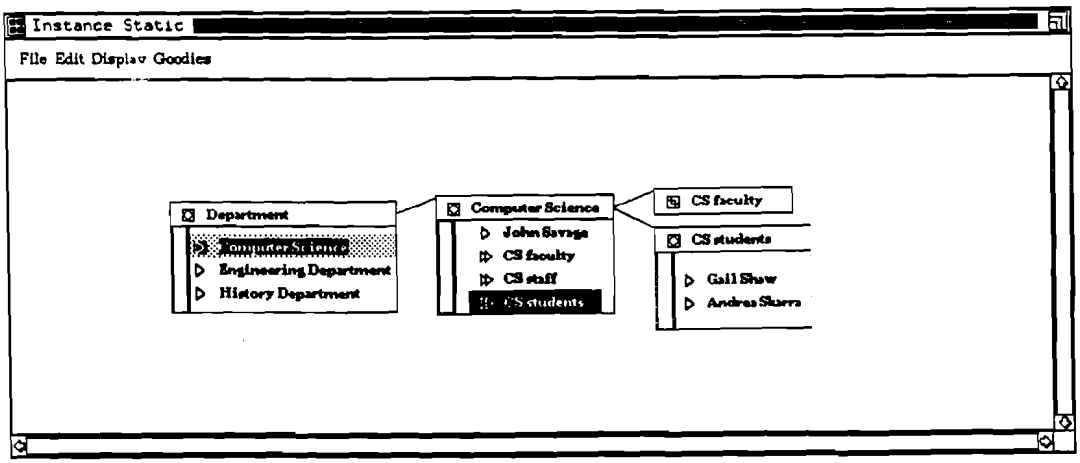
(b) Multiple chains

Figure 6. Instance Browser

ognize the reference relations among instances in a chain. Figure 6(a) shows that there is a chain of references from *ComputerScience* to *Advises* through *CSfaculty* and *StanZdonik*. It is very useful because displaying many objects without tracing will certainly confuses the user. Smalltalk interface has similar mechanism for series of related objects.

More than one chain of instances can be displayed when the user executes **NewChain** on a selected instance. Figure 6 (b) shows such two chains: One started from *AssociateProfessor* and the other from *AssistantProfessor*. Deleting an instance object is done by **RemoveObj**. As mentioned in Type Browser, the selected instance and instances following it are removed from display. A similar effect is achieved by **Simplify** with the difference that the selected instance is not removed, but is simplified, and following instances are temporarily invisible. **DisplayFront** will expand the selected simple object to its full view and all the instances following are redisplayed as they were before they were removed. Since **FollowChain** only displays one full view while collapsing all others, it saves window space. Figure 6(c) shows the display after the user followed from *CSstudent*. *CSfaculty* object is now collapsed and following instances are removed from the display. However, the user can display more than one full view of instances expanded from the same instance at the same time by **DisplayFull**. That is, the selected instance is expanded to its full view while current full view instance also remains the same. This feature is useful since the user may want to compare two instances.

Currently, GUI does not provide automatic update of an object across the different windows. If there is any change in the status of an instance outside of the Instance Browser, the change does not automatically trigger update of display. It is a responsibility of the user to execute periodically **UpdateInstance** so that it is reflected in the Instance Browser if there was any change. Automatic update of different views of a same object is certainly required and further work is necessary in the future. Figure 7 shows different windows of GUI opened during a typical session.



(c) Simplification of CSfaculty object

Figure 6. Instance Browser

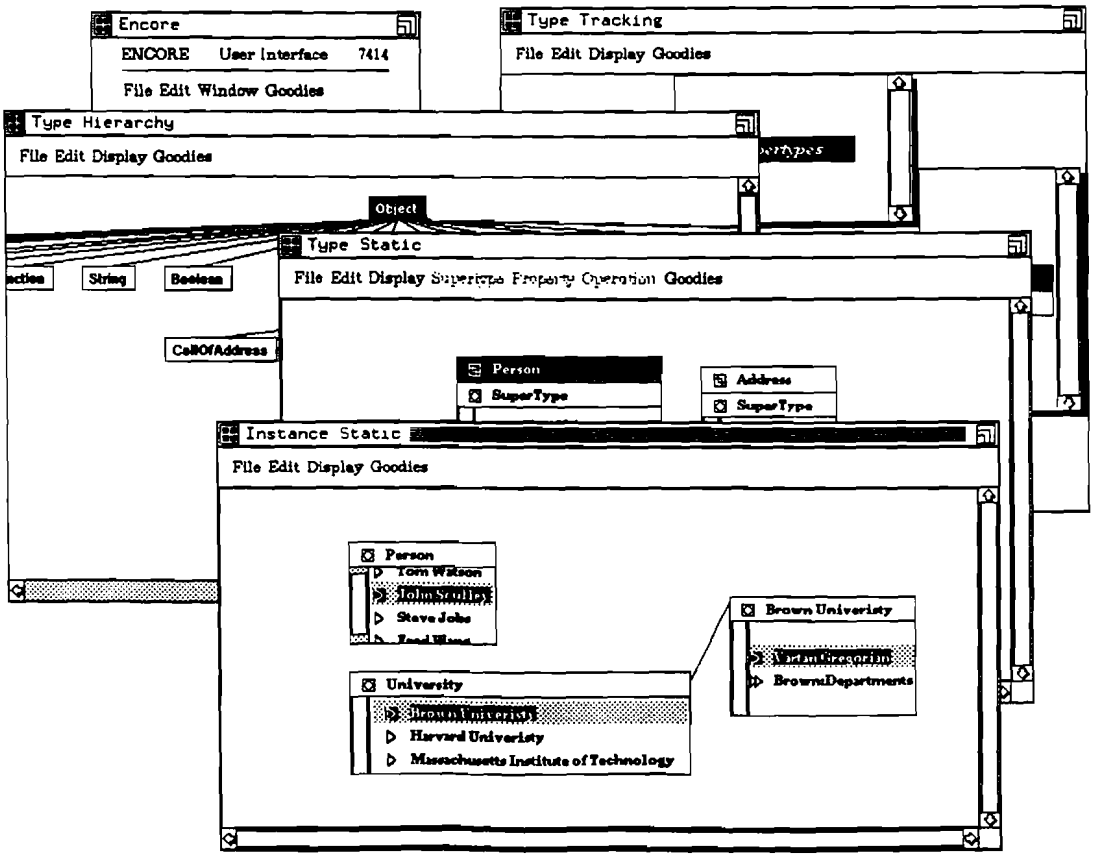


Figure 7. Overlapping GUIwindows during a session

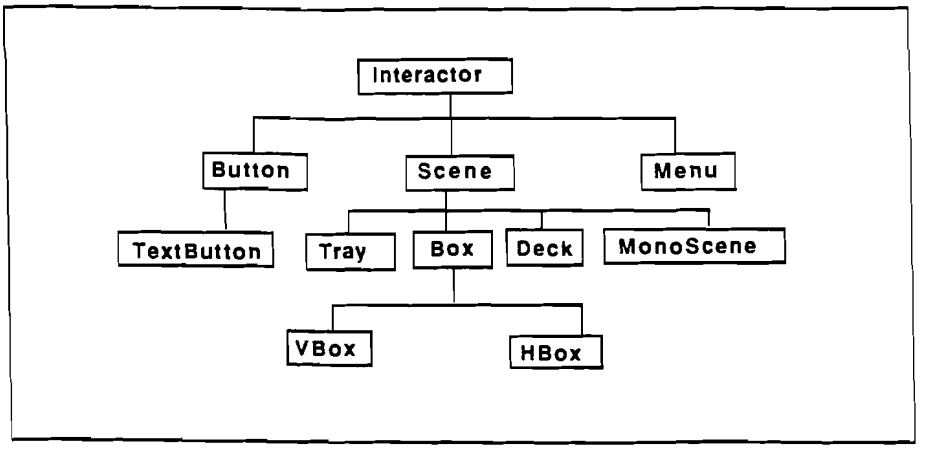
3 Implementation

GUI is implemented using the object oriented graphical user interface package **InterViews**[10]. InterViews is written in C++ and runs on top of the X window system. We chose InterViews as implementation package because GUI is naturally object-oriented, many of its predefined classes can be easily used and it provides a derivation of subclasses. Also, InterViews has relatively shallow class hierarchy, which is desirable feature for quick prototyping since we can easily understand the relationships of classes. If levels of a class hierarchy are too deep, understanding relationships among them may not be easy. InterViews is not really as low level a toolkit as X, but neither very high level package. Thus, InterViews provides enough support for user interface design while it does not restrict the freedom of interface design too much. There are two major classes in InterViews: Interactor is the base class for all interactive objects from which the user interface is built, Scene class is the base class for composite object and is a subclass of the Interactor. Classes defined for the **Type Browser** and **Instance Browser** are derived from a subclass of the Scene class. In the following section, I will describe the overall hierarchy of classes defined in GUI in relation to InterViews' classes. Figure 8(a) shows the hierarchy of InterViews' classes from which GUI classes implementing the browser are derived. Figure 8(b) shows the hierarchy of GUI classes with InterViews. Rectangles with italicized strings represent the GUI classes for Browser while others are from InterViews.

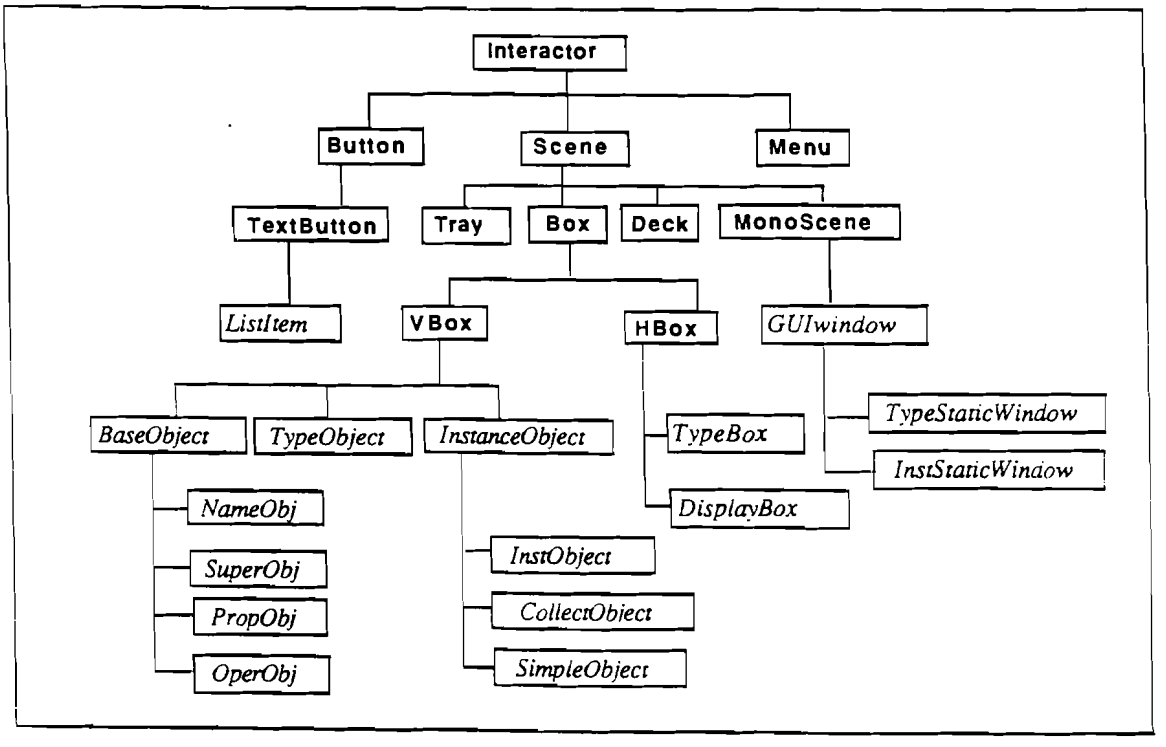
3.1 Class Hierarchy

This Section will describe the hierarchy of classes defined for the **Type Browser** and **Instance Browser**, relating them to existing InterViews classes.

The Scene class has 3 major subclasses: Box, Tray, and Deck. The Box class is a composite object that tiles it's components while Tray overlaps its objects. The Deck class provides a composition of objects on top of each other, where only the top object is visible. The Deck class does not provide an appropriate layout for a browser even though it is very useful in multiple page text layout. Our main classes **TypeStaticWindow, InstStaticWindow, TypeObject, BaseObject,** and **InstanceObject** are all derived from the Box



(a) Hierarachy of InterViews classes from which classes of GUI browser are derived



(b) Hierarchy of Classes of GUI browser and InterViews

Figure 8. Class hierarchy of InterViews and GUI browser

classes.

TypeStaticWindow and **InstStaticWindow** are used for laying out the type and instance objects of ENCORE and provide data and operations for manipulating those objects. **TypeObject, BaseObject**, and **InstObject** are classes for display objects corresponding to types and instances. **TypeObject** is merely a composite object of instances of BaseObject. A TypeObject consists of 4 subobjects, which are subclasses of BaseObject: **NameObj, SuperObj, PropObj** and **OperObj**. InstObject also has 3 subclasses, **InstanceObject, CollectObject**, and **SimpleObject**. InstanceObject is a class for displaying objects of single complex type while CollectObject is a class for displaying objects of type ColType. SimpleObject is a class for different view of both InstanceObject and CollectObject. This class is added for saving display space when many objects are displayed simultaneously. Since there is no class for stacking objects where top object is fully visible while the other objects shows a part of full view, we collapse an object (type or instance) when it is not of current interest so that available space is allocated for other object's display.

3.2 TypeStaticWindow and InstStaticWindow Class

Type Browser is implemented using **TypeStaticWindow** class. A type object is an instance of the **TypeObject** class. **TypeObject** is derived from the **VBox** class of InterViews and is a class for composite objects. **BaseObject** is also derived from the **VBox** because it implements subobjects of a type object and each of these subobjects tiles its components, instances of **ListItem**, vertically. It is a base class for classes for components of a type object. **NameObj** is the class for name component of a type and **SuperObj** is the class for supertype components. Property and operation components are created from **PropObj** and **OperObj** class, respectively. A simplified object is implemented by **SimpleObj** class. SimpleObj is a vertical box containing one TextButton, which is used to display the name of a type. Thus, when a type object is simplified, **TypeObject** only contains **SimpleObj** as components. Each element of subcomponents is again an object and each is implemented using class ListItem, which is a subclass of TextButton class from InterViews. Thus, when the user selects an item from one of those subcomponents, it is highlighted using

inverse video. Currently, 2 supertypes, 4 properties and operations each are displayed in full view of a type object. A scrollbar is used to examine the rest of the elements. InterViews provides a built in scrollbar object.

Since the **TypeStaticWindow** is subclass of VBox, more than one chain of type objects can be displayed, vertically. Internally, the type browser is composed of number of horizontal boxes vertically tiled with each other. Each chain is displayed in one of the horizontal boxes. In turn, each of horizontal box is composed of vertical boxes tiled horizontally. Thus, each type object along a chain is displayed in one of these inner vertical boxes.

As with TypeStaticWindow, the **InstStaticWindow** is also a subclass of **VBox** class of InterViews. **InstStaticWindow** class is used to layout the instances and provides the operations to manipulate on instances. Each instance object has two views. One is a full view and is implemented by **InstObject**. The other one is a simplified view and is implemented by **SimpleObject**. An instance can be either an instance of simple or complex type or an instance of a collection type. An instance of a collection type is in turn a list of instances and it is implemented using **CollectObject** class. All three classes, InstObject, CollectObject, and SimpleObject, are derived from **InstanceObject** class, which is a subclass of **VBox** class.

Since the size of an instance object determines the number of objects that can be displayed in the Instance Browser, determining a reasonable size for an instance is very important for a display. Currently, the height of an object of **InstObject** class is limited to the height of 5 **TextButton** objects. The size of components of **Box** object is changed when the size of a containing object is changed because components are glued together by a **Glue** object. However, we felt that it is not desirable to change the size of type or instance objects when a window is resized because the change of size produces an undesirable shape.

4 Future Work

This Section will mention future directions for this project. In particular, Schema design and Instance creation will be mentioned.

GUI should provide different levels of the interface for different users. Even

though our interface was intended for mainly novice users, it would be desirable to have different levels for those who are more advanced. Hiding or providing different features at different levels could be considered.

As mentioned, GUI currently does not provide complete facilities for schema design. Partial work is done in this area and but it is more textual than graphical. Further work is necessary for the design review and the implementation. Separate windows for type and instance creation are desirable. Since any user types are subtypes of existing types, creation of types is basically a modification of a template of a supertype of the type to be created. The system should provide an way to move objects so that the user can supply definitions graphically. Likewise, the user can create an instance graphically by moving actual display objects, or assigning edges between properties and values in addition to the current facility.

Since the performance of GUI was not primary consideration in building the prototype, it does not meet the speed expectations of the user. However, this should be resolved in near future to meet the end user's expectation.

It is often confusing when many different windows are displayed simultaneously and the windows contains different views of the same object. GUI should provide a mechanism to reflect any changes in an object across all windows; Now one window could show an object A while the object A had been deleted from other window.

It is often the case that too many objects are displayed at once which may over burden the user. It would be nice if the user could hide unrelated or unimportant display objects if they are not of current interest. As in Type Hierarchy window, a feature for localizing a chain of objects can be added for the purpose of hiding non-focal objects.

Zooming on a view of the Type and Instance browser windows is another feature which could be added. Currently, the user adjusts the view using vertical and horizontal scrollbars. However, this does not change the view of an object itself, but shows different portions of a window. Zooming is a good feature in providing the different levels of views of objects.

Finally, customization of a display of objects and an object itself can be added as a browsing feature. This implies also the capability of moving graphical objects, which allows easier comparison among the objects.

5 Conclusion

Overall functionality of GUI, particularly, Type and Instance browser was described in this report. Since databases contain huge amount of data and their schemas are very complex, some kind of browsing facility is essential. With advances of window systems ,high resolution bitmapped displays, and pointing devices, graphical interfaces to databases become an increasingly interesting field. However, many interfaces fall short in meeting the users' expectation. Some fail because they do not provide a suitable browsing facility. Others have a global hierarchical display of types that does not capture the entire schema of the underlying database and the schema display is disconnected from instance display. Some fail to group coherent or related objects for comparison purposes. Therefore, they do often fail to utilize the strength of a graphical interface to provide an efficient and convenient way to manipulate data.

Our browser provides a convenient way to browse data at both the schema and instance level by simply following chains, and it provides a consistent view across levels. With some improvements in the areas mentioned above, GUI will be a convenient and efficient tool to access the database and manipulate data.

6 Acknowledgment

I would like to thank Stan Zdonik for his direction in the project. I also thank Page Elmore for her guidance and helpful advice in the development of project and the preparation of this paper. Finally, My thanks go to Greg Brail and Hisato Kato for implementing several parts of our graphical interface.

7 References

1. D.Bryce em and R.Hull, **SNAP: A Graphics- Based Schema Manager**, *IEEE conference on Data Engineering*, Los Angeles, CA, February 1896.

2. Davidson,J.W. em and S.B.Zdonik, **A Visual Interface for a Database with Version Management**, *Trans. OIS*, 4(3):226-242, 1986.

3. Dennis Fogg **Lesson From a "Living in a Database" - Graphical Query Interface**, *ACM SIGMOD*, 14,2, Proceedings of Annual Meeting, June 18-21, 1984.

4. Goldman,K,J., S.A.Goldman, P.C.Kanellakis and S.B.Zdonik **ISIS: Interface for a Semantic Information System**, *International Conference on Management of Data*, 1985.

5. Hisato Kato, **Graphical User Interface to Object-Oriented Database**, *Brown University Master's Project Report*, 1989.

6. Herot, C.F, **Spatial Management of Data**, *TODS*, 5(4): 493-513, 1980.

7. Kim, H-J., H.F.Korth, and A.Silberschatz, **PICASSO: A Graphical Query Language**, *Software - Practice and Experience*, 18(3):169-203,1988.

8. Harry K.T.Wong and Ivy kuo, **GUIDE: Graphical User Interface for Database Exploration**, *Proc. Eighth VLDB*, 22-32, Mexico City, Sept. 1982.

9. J.A.Larson, **A Visual Approach to Browsing in a Database Environment**, *IEEE Computer*, 62-70, 1986.

10. Linton,M.A., Vlissides and P.R.Calder, **Composing User Interfaces with InterViews**, *IEEE Computer*, 22(2):8-22,1989.

11. D.Maier, P.Nordquuist and M.Calder, **Displaying Database Objects**, *Expert Database Systems*, 59-73, 1986.

12. Shaw,G.M. and S.B.Zdonik, **A Query Algebra for Object-Oriented Databases**, *Brown University Tech Report CS-89-19*, March 1989.

13. Skarra,A.H, S.B.Zdonik and S.P.Reiss, **Observer:An Object Server for an Object Oriented Database System**, *Brown University Tech Report CS-88-08*, July, 1987.

14. Stonbreaker,M. and J.Kalash, **TIMBER:A Sophisticated Relational Browser**, *Eighth International Conference on Very Large Data Bases*, 1-10, 1982.

15. Zdonik,S.B. and P.Wegner, **Language and Methodology for Object Oriented Database Environments**, *Nineteenth Annual International Conference on System Sciences*, 1986.

16. M.M.Zloof, **Query by Example**, *Proceedings of the National Computer Conference*, May 1982.

Graphical User Interface to ENCORE

— Specification for Type and Instance Browser

Young Woo Koh

Department of Computer Science
Brown University

December, 1989

Contents

Documentation for Type and Instance Browser

Young W. Koh

December 21, 1989

1 Introduction

This document describes the classes used in implementing **Type Browser** and **Instance Browser** of **GUI**. **TypeStaticWindow** class provides operations to manipulate type objects and **InstStaticWindow** is the class for manipulation of instance objects. **TypeObject** and **InstanceObject** are the classes to construct the types and instance objects for display. Since the type and instance objects are made of smaller objects of different classes, both **TypeObject** and **InstanceObject** composes those smaller objects. **ListItem** is the class that creates the smallest element of both type and instance object. In the following, Each of these classes will be described in detail.

2 Type Browser

This section describes the classes used for implementation of **Type Browser**. **TypeStaticWindow** is a subclass of **GUIwindow**, which is ,in turn, a subclass of InterViews's **Scene** class. It provides data and operations to manipulate the displays of ENCORE type objects so that user can browse the schemas of ENCORE graphically. Construction of actual display objects corresponding to ENCORE type objects is accomplished by uses of **TypeObject** *and* **BaseObject** classes, which are explained in following subsections.

2.1 Class TypeStaticWindow

TypeStaticWindow class is a subclass of *GUIwindow* class, which is the base class for all the window classes in GUI.

Superclass : GUIwindow
Instance Variables :

TypeBox* : thebox
The outer most box object contains all the displays of the type objects in the Type Browser.
TypeObject* : curtypeobj
Currently selected type object.
void* : cursubobj
selected subobjects of the current type object. The subobjects is one of name, property, supertype, or operation object.
ButtonState* : typestate
The buttonstate for the **Type Browser**.
TypesNode* : curtypenode
The currently selected typenode, which contains the information

about the type object.

GUIbaseNode* : curitemnode
Currently selected item node, which contains the information about the current item.

ListItem * : curitem
Currently selected item, which is the smallest component of a type object.

ObjectList* : boxlist
The list of boxes for a given level.

ObjectList* : typelist
The list of type objects for a given box.

boolean : nameobject
Indicates whether the name object is displayed or not for a type object is displayed or not for a type object.

boolean : superobject
Indicates whether the supertype object is displayed or not for a type object.

boolean : operobject
Indicates whether the operation object is displayed or not for a type object

boolean : parmobject
Indicates whether the parameter object is displayed or not for a type object

boolean : itemobject
Indicates whether the item object is displayed or not for a type object

VBox * : dummyinnerbox
Dummy inner box containing no object. This is needed to hold the space when all the objects in the box are removed.

VBox * : dummyinnerbox
Dummy outer box containing no object. This is needed to hold the space when all the objects in the box are removed.

MenuActivator* : propertyMenu
Menuactivator for property menus.

MenuActivator* : supertypeMenu
Menuactivator for supertype menus.

MenuActivator* : operationMenu
Menuactivator for operation menus.

*** Internal Operation

void
ClearBoxes (TypesNode* typenode)

Effects : This routine clears boxes that contains type objects to be removed from the display because the instance object of **typenode** is simplified. The objects to be removed are chained from the instnode. The information for those objects are not deleted because they are temporarily removed from the screen.

void
DeleteBoxes (TypesNode* typenode)

Effects : his routine deletes boxes that contains type objects to be removed from the display. The objects to be removed are chained from the instnode. The information of deleted objects is also removed from the database of browser.

void
DispNormal ()

Effects : This routine display all the items that were dimmed to their normal state.

void
DisplayParameter ()

Effects : This routine displays the list of parameters for a selected operation from a type object. The list of parameters are displayed in a rectangular box as with any other type object except that it only has one subcomponent, which is the list of parameters.

void
DisplaySuperRefer (ObjectList* superlist)

Effects : This routine displays all the objects referred from the elements of supertype component of a type object. All those objects will be displayed as exactly same as before they were removed from display.

void
DisplayPropRefer (ObjectList* proplist)

Effects : This routine displays all the objects referred from the elements of property component of a type object. All those objects will be displayed as exactly same as before they were removed from display.

void
DisplayOperRefer (ObjectList* operlist)

Effects : This routine displays all the objects referred from the elements of operation component of a type object. All those objects will be displayed as exactly same as before they were removed from display.

void
DisplayParmReferRefer (ObjectList* parmlist)

Effects : This routine displays all the objects referred from the elements of parameter component of a type object. All those objects will be displayed as exactly same as before they were removed from display.

void
ExpandTitle (TypesNode* typenode)

Effects : This routine expands a simplified type object to its intermediate form, which only shows the titles of it components. Each title can be expanded further.

void
ExpandAll (TypesNode* typenode)

Effects : This routine expands a simplified trype object to its full view where all the components are shown.

void
ExpandProperty (TypesNode* typenode)

Effects : This routine expands a property portion of a type object if it was simplified.

void
ExpandSuperType (TypesNode* typenode)

Effects : This routine expands a property portion of a type object if it was simplified.

void
ExpandOperation (TypesNode* typenode)

Effects : This routine expands an operation portion of a type object if it was simplified.

void
ExpandParameter (TypesNode* typenode)

Effects : This routine expands a parameter portion of a type object if it was simplified.

void
FindBoxes (TypesNode* typenode)

Effects : This routine assigns a box object that will contain the object in typenode to the next box object following a box that conatins curnode. If the box is dummy box, it will create box and assign it to typenode.

void
Init()

Effects : This routine initializes the instance variables to their initial values. **Init()** of **GUIwindow** is called to initializes the instance variables inherited from it because **TypeStaticWindow** is the subclass of **GUIwindow** .

void
InitMenus ()

Effects : This routine constructs the menus for the browser. It inherits common menus such as **File** *and* **Edit** from **GUIwindow**. **Display, Supertype, Property,** *and* **Operation** menus are created for the Type Browser.

void
MakeItSimple (TypesNode* typenode, ItemType itype)

Effects : This routine simplifies the type object to its simple object depending on the itemtype, **itype**. If itype is **NameTitle**, it will simplify whole type object to its simple form. If itype is **SuperTitle**, it will simplify the supertype portion of the type object to its simple form.

TypesNode*
NewTypeNode (TypeObject* typeobject)

Effects : It returns a TypesNode that contains the information about the type object to be displayed. **TypesNode** is used to stores the information about the display objects corresponding to ENCORE type objects displayed in the Type Browser.

void
NewChain ()

Effects : This routine creates a new chain to display new type object selected. New chain starts right below the last chain and newly displayed type object becomes the head of the chain. The chain is placed in a **HBox** of InterViews and type objects belonged to that chain become the components of the **HBox**.

void
InnerBoxExist (TypesNode* typesnode)

Effects : This routine checks if there are other type objects referred from the same type object that this new type object was referred from. If so, it will place the new type object at the bottom of the same **VBox** where those objects are placed. **typenode** contains information about the new type object to be displayed.

void
OuterBoxExist (TypesNode* typesnode)

Effects : This routine creates a space (VBox) where this new type object is displayed. Since there was no previous type objects that have the same parent with this new one, this routine creates a space for the group. Subsequently, any new object referred from same parent will be displayed at the same box just created.

void
NoBoxExist (TypesNode* typesnode)

Effects : This routine creates both outer box and inner box because there is no objects displayed at the same level that this new type object is to be displayed. A level is determined by number of objects displayed along horizontal box. For example, if an object A is at level 1, then any objects referred from A are displayed at the level 2.

void
MakeItFront ()

Effects : This routine displays the full view of typenode and simplifies any fully displayed objects which are referred from same type object.

void
Remove (TypesNode* typenode, boolean flag)

Effects : This routine will either remove a type object permanently or temporarily remove from the display. If the **flag** is **true**, it will remove the type object both from the display and display list. Otherwise, it is removed from display.

void
RemoveObject ()

Effects : This routine removes a type object selected.

void
SimplifyAll (TypesNode* typenode)

Effects : This routine will simplify a type object so that only the name of type is shown. All the objects following that type will be removed from the display, temporarily.

void
SimplifyProperty (TypesNode* typenode)

Effects : This routine simplifies the property portion of type object, typenode, so that only the title is shown.

void
SimplifySuperType (TypesNode* typenode)

Effects : This routine simplifies the supertype portion of type object, typenode, so that only the title is shown.

void
SimplifyOperation (TypesNode* typenode)

Effects : This routine simplifies the operation portion of type object, typenode, so that only the title is shown.

void
SimplifyParameter (TypesNode* typenode)

Effects : This routine simplifies the parameter portion of type object so that only the title is shown.

void
SimplifySuperRefer (ObjectList* superlist, boolean flag)

Effects : This routine is called when the supertype portion of a type object is simplified. All the object following the simplified one will be removed from display. It simplifies those objects referred from each element of supertype object of the type.

void
SimplifyPropRefer (ObjectList* proplist, boolean flag)

Effects : This routine is called when the property portion of a type object is simplified. All the object following the simplified one will be removed from display. It simplifies those objects referred from each element of property object of the type.

void
SimplifyOperRefer (ObjectList* operlist, boolean flag)

Effects : This routine is called when the operation portion of a type object is simplified. All the object following the simplified one will be removed from display. It simplifies those objects referred from each element of operation object of the type.

void
SimplifyParmRefer (ObjectList* parmlist, boolean flag)

Effects : This routine is called when the parameter portion of a type object is simplified. All the object following the simplified one will be removed from display. It simplifies those objects referred from each element of parameter object of the type.

void
ShowProperty ()

Effects : This routine shows all the properties inherited from a supertype selected by dimming out all other properties in the type definition.

void
ShowOperation ()

Effects : This routine shows all the **operations** inherited from a supertype selected by dimming out all other operations in the type definition.

void
ShowAll ()

Effects : This routine shows all the **operations** and **properties** inherited from a supertype selected by dimming out all other properties and operations of the type object.

void
TogglesSrcWindow ()

Effects : This routine opens the **source window** if it is not already opened. The window s empty. If the window is already opened, it will close the window.

void
UpdateSrcWindow ()

Effects : This routine updates the contents of the source window to the contents of newly selected operation object if the source window is already opened. Otherwise, it opens the window first and displays the contents of the operation.

*** Protected Operation

virtual void
Reconfig ()

Effects : This routine recalculates browser window's configuration when there are any changes.

virtual void
Resize ()

Effects : This routine is called when the window's size is modified.

virtual void
Draw ()

Effects : This routine redraws the contents of the window when there are changes such as view change or damage on the window.

virtual void
Copy ()

Effects : This routine copies the any strings selected and puts in the clipboard window.

virtual void
Handle (Event& events)

Effects : This routine receives events from either from a keyboard or mouse and invokes appropriate operations.

*** Public Operation

TypeStaticWindow (GUIappl* parent)

Effects : This routines creates a TypeStaticWindow by calling Init and InitMenus routine. **parent** is the instance of **GUIwindow.**

TypeBox*
GetBox ()

Effects : This routine returns the outer most box that contains all the display objects of the Type Browser.

void
DisableExtraMenus ()

Effects : This routine disables all the extra menu which are not needed at given moment when an object is selected.

void
DisplayType ()

Effects : This routine will display the type object, typenode. Depending on the status of type object, it will construct and display the selected type object and all the other type objects following that type object.

void
EnableOper ()

Effects : This routine enables the menu items for manipulation of **operation** objects. This is called when an operation object is selected in most time.

void
EnableProp ()

Effects : This routine enables the menu items for manipulation of **property** objects. This routine is called when a property object is selected.

void
EnableSuper ()

Effects : This routine enables the menu items for manipulation of **operation** objects. This routine is called when a supertype object is selected.

void
Expand ()

Effects : This routine expands the type object that previously simplified to its intermediate view. The intermediate view shows only the title of its subcomponents. Each of subcomponent can be further expanded to show the contents of it.

void
ExpandType ()

Effects : This routine expands a selected type object. The user can select whole type object or portion of it, then it will expand the selected portion to its full view.

virtual void
Invoke ()

Effects : This routine invokes an operation on selected object.

void
NewDef (ENObject* enobject, ModificationType mtype)

Effects : This routine displays new type object which was selected from outside of the **Type Browser.**

void
NewType (Type* type)

Effects : This routine searches list of type objects being displayed to find if there is a type, **type.** If it exist, it replaces the old one with newly constructed one of **type.**

void
SetCurObject (TypeObject* curobject)

Effects : This routine sets **curobject** to the currently selected type object.

virtual void
ShowType ()

Effects : This routine displays a full view of the type object next to the object referred if the new object is not already displayed. If the object exists, it will highlight the name of the existing type object.

void
Simplify ()

Effects : This routine simplifies the selected object if the object is not already simplified. The simplified object displays only the name of the type. If there are type objects following this object, all following objects are removed and an arrow is displayed at the simplified one to indicate further chain of objects.

virtual void
ShowInst ()

Effects : This routine displays a list of instances of the selected type object in the **Instance Browser.** If instance browser window is not opened, it will open the window first.

2.2 Class TypeObject

This section describes the class **TypeObject.** It is a subclass of **VBox** of InterViews. It constructs a display type object corresponding to ENCORE types by composing the subcomponents. A type object is composed of **Name, SuperType, Property,** *and* **Operation** objects. Thus, this class provides the data and operations for composing and manipulating those sub objects.

SuperClass : VBox
Instance Variables :

TypeStaticWindow* : typewindow
TypeStaticWindow that this type object is displayed.
TypesNode* : typenode
TypesNode that contains this type object.
Type* : type
ENCORE type for the type object.
ButtonState* : state

Buttonstate for the type object.

Command : pop
Command for expanding simple object to full view.

Command : shrink
Command to shrink the type object to simplified one.

Command : follow
Command to display the contents of the selected item(type).

ObjectList* : proplist
List of properties belong to the type.

ObjectList* : operlist
List of operations belong to the type.

ObjectList* : superlist
List of supertypes belong to the type.

ObjectList* : parmlist
List of parameters belong to the type.

PropObj* : propobject
Property object of the type.

OperObj* : operobject
Operation object of the type.

SuperObj* : superobject
Supertype object of the type.

PropObj* : parmobject
Parameter object of the type.

NameObj* : nameobject
Name object of the type.

boolean * : titleitem
Indicates whether the name object is selected.

boolean * : arrow
Indicates whether the type has following chains from it.

void* : cursubobj
Currently selected subobject of the type.

ListItem * : curitem
Currently selected item of the type.

ListType : listtype
Indicates which part of the type is being displayed.

NameNode* : namenode
Namenode that contains an item object of the Name object.

NameNode* : supernode
Namenode that contains an item object of the Supertype object.

NameNode* : propnode
Namenode that contains an item object of the Property object.

NameNode* : opernode
Namenode that contains an item object of the Operation object.

NameNode* : parmnode
Namenode that contains an item object of the Parameter object.

int : width
Width of the box containing the type.

int : itemwidth
Width of the box containing an individual item of the type.

int : itemheight
Height of the box containing an individual item of the type.

*** Internal Operation

void
Init ()

Effects : This routine initializes the instance variables of the type object.

void
MakeList (Type* type)

Effects : This routine calls routines that makes the list of for the subobjects of the type object. Subobjects are **Name, property, Supertype,** *and* **Operation** object.

void
MakeParmList (GUIobject* operobj)

Effects : This routine forms list of parameters for an operation object, **operobj** of the type object.

void
MakeObject (Type* type)

Effects : This routine constructs the type object of given type ,**type** by composing subobjects.

void
MakeParmObject ()

Effects : This routine forms **Parameter** object for the selected operation object. The list of parameters is formed by **MakeParmList.**

void
MakeTitleObject (Type* type)

Effects : This routine makes the title object of each subobject. Title object is the display object indicating subobjects of the type object. That is, it constructs titlebars for each subcomponent of the type.

void
MakePropList (Type* type)

Effects : This routine forms a list of properties of the type object given in **type.**

void
MakeSuperList (Type * type)

Effects : This routine forms a list of supertypes of the type object given in **type.**

void
MakeOperList (Type* type)

Effects : This routine forms a list of operations of the type object given in **type.**

ObjectList *
InheritPropList (Type* supertype)

Effects : This routine forms and returns a list of properties of the type that are inherited from **supertype.**

ObjectList *
InheritOperList (Type* supertype)

Effects : This routine forms and returns a list of operations of the type that are inherited from **supertype.**

NameObj*
MakeNameObj (Type* type)

Effects : This routine makes a name object of the type, **type.**

SuperObj*
MakeSuperObj (ObjectList* superlist)

Effects : This routine returns a supertype object of the type constructed from the supertypes in **superlist.**

PropObj *
MakePropObj (ObjectList* proplist)

Effects : This routine returns a property object of the type constructed from the properties in **proplist.**

OperObj *
MakeOperObj (ObjectList* operlist)

Effects : This routine returns an operation object of the type constructed from the operations in **operlist.**

*** Protected Operation

void
Reconfig ()

Effects : This routine reconfigures the type object when there is any change in the type object.

*** Public Operation

TypeObject (GUIobject* guiobj, ButtonState* state, ListType ltype, TypeStaticWindow* typewindow, Command pop, Command shrink, Command follow, boolean parmflag)

Effects : This constructor creates a type object when the type is first displayed. **guiobj** is a GUIobject corresponding to this type object. **state** is a buttonstate and **ltype** is list type which determines what portion to be displayed. **typewindow** is TypeStaticWindow where this type is displayed. **pop, shrink,** *and* **follow** are commands for expanding, shrinking, and newly displaying of the type object. **parmflag** indicates whether the type is referred from a parameter object.

TypeObject (TypesNode* typenode, ButtonState* state, List Type ltype, TypeStaticWindow* typewindow, Command pop, Command shrink, Command follow, boolean parmflag)

Effects : This routine constructs the type object using the information stored in **typesnode.** This constructor is used to rebuild the type objects that were temporarily removed from the browser. **typenode** contains information for the type to be displayed.

TypeObject (TypesNode* typenode, ButtonState* state, List Type ltype, TypeStaticWindow* typewindow, Command pop, Command shrink)

Effects : This constructs an special type object, which is the list of parameters of an operation object. Thus, the type object has only one subcomponent, which is a list of parameter.

~TypeObject ()

Effects : Destructor of **TypeObject**.

void
SetCurObject (ListItem*)

Effects : This routine sets the selected item to the currently selected item. The selected item is an instance of **ListItem**.

TypesNode*
GetTypesNode ()

Effects : This returns the TypesNode that contains currently selected type object.

ListType
GetListType ()

Effects : This returns the type of the display of the type object, which determines the portion of the type to be displayed. That is, the type may be displayed with only the list of properties or operations of the type, instead of full view.

NameObj*
GetNameObject ()

Effects : This routine returns the name object of the type.

PropObj*
GetPropObject ()

Effects : This routine returns the property object of the type.

SuperObj*
GetSuperObject()

Effects : This routine returns the supertype object of the type.

OperObj*
GetOperObject ()

Effects : This routine returns the operation object of the type.

ObjectList*
GetSuperList ()

Effects : This routine returns the list of the supertypes of the type.

ObjectList*
GetPropList ()

Effects : This routine returns the list of the properties of the type.

ObjectList*
GetOperList ()

Effects : This routine returns the list of the operations of the type.

ObjectList*
GetParmList ()

Effects : This routine returns the list of the parameters of an operation object of the type.

TypeStaticWindow*
GetTypeStat ()

Effects : This routine returns the **TypeStaticWindow** where this type is being displayed.

void
SetCurItem (ListItem* curitem)

Effects : This routine sets **curitem** to the currently selected item of the type.

ListItem*
GetCurItem ()

Effects : This routine returns the currently selected item of the type.

void
GetCoord (Coord& xcoord , Coord& ycoord)

Effects : This routine returns x and y coordinate at the right top corner of the box forming the type object. X and y coordinates are stored in **xcoord** and **ycoord**, respectively.

void
GetItemCoord (Coord& xcoord , Coord& ycoord)

Effects : This routine returns x and y coordinate at the left top corner of the box forming a type object. X and y coordinates are stored in **xcoord** and **ycoord**, respectively.

2.3 Class BaseObject

The **BaseObject** is a subclass of **VBox** of InterViews, which is an interactor object that tiles it components. **BaseObject** is the superclass for the **NameObj, PropObj, OperObj** and **SuperObj** and it provides data and operations common to its subclasses. **NameObj** is the class for name objects. **PropObj** is the class for Property objects. **SuperObj** is the class for Supertype objects. Finally, **OperObj** is the class for Operation objects.

SuperClass : VBox
Instance Variables :

ListItem* : nameitem
Title object of the base object.
IconType* : icontype
Icontype of this base object.
ClearPort* : viewport
Clearport where this object is drawn.
TypeObject* : typeobject

Type object that this base object is belonged to.

ButtonState* : state
Buttonstate for the type object.

Command : pop
Command for expanding simple object to full view.

Command : shrink
Command to shrink the type object to simplified one.

Command : follow
Command to display the contents of the selected item(type).

int : width
Width of the box containing the type.

int : itemwidth
Width of the box containing an individual item of the type.

int : itemheight
Height of the box containing an individual item of the type.

*** Internal Operation

Interactor*
ScrollFrame ()

Effects : This routine returns the vertical scroller for this object. The scrollbar is used to scrolls the contents of the baseobject, which is a list of properties or operation because size of the baseobject is fixed while the list can be quite large.

void
Reconfig ()

Effects : This routine reconfigures the base object when there is any change in the size of the box forming the base object.

void
MakeItemBox ()

Effects : This routine constructs a box containing the items belong to this object. Each item is an instance of **ListItem**.

*** Protected Operation

virtual void
MakeObject (ItemType itype)

Effects : This routine composes subcomponents of baseobject to constructs a baseobject, whose items are of **itype**. That is, if the itype is property, then it constructs a property object portion of a type object.

*** Public Operation

BaseObject (ButtonState* state, TypeObject* typeobject, Type StaticWindow* typewindow, Command pop, Command shrink)

Effects : Constructor of BaseObject. **state** is the button state and **typeobject** is the type object where this base object is belonged to. **typewindow** is the TypeStaticWindow where **typeobject** is to be displayed. **pop** *and* **shrink** are commands for expanding and simplifying the baseobject, respectively.

void
SetCurItem (ListItem* curitem)

Effects : This routine sets **curitem** to the currently selected item of the baseobject.

ListItem*
GetCurItem ()

Effects : This routine returns the currently selected item of the base object.

TypeObject*
GetTypeObject ()

Effects : This routine returns the **TypeObject** to which this base object is belonged to.

int
GetWidth ()

Effects : This routine returns the width of the base object. The width of the object is the width of the widest item of the base object.

VBox*
GetItemBox ()

Effects : This routine returns the box forming the base object.

void
SetItemWidth (int width)

Effects : This routine sets the width of the box to **width**.

ListItem*
GetNameItem ()

Effects : This routine returns the item that representing name object of the base object.

2.4 NameObj

This section describes **NameObj** class which implements the name object portion of a type object.

SuperClass : BaseObject
Instance Variables :

Type* : type
ENCORE type.
boolean : arrowflag
Flag indicating existence of an arrow.

*** Public Operations

NameObj (Type* type, char* name, ButtonState* state, TypeObject typeobj, TypeStaticWindow* twindow, Command pop, Command shrink, ItemType itemtype, boolean commandflag, boolean arrowflag)

Effects : Constructor of Name object of a type object. **type** is ENCORE type of object and **name** is string name of the type. **state** is a button state of a type object. **typeobj** is a type object where this name object is belong to. **twindow** is TypeStaticWindow where the type object is displayed. **itemtype** is type of ListItem for this object. **pop, shrink,** and **follow** are Commands for expanding, shrinking, and displaying full view of the type definition, respectively.

2.5 SuperObj

This section describes **SuperObj** class which implements the supertype objects of a type object.

SuperClass : BaseObject
Instance Variables :

ObjectList* : superlist
List of supertypes of a type object.
ListItem* : curitem
A supertype item currently selected.

*** Internal Operations

void Init ()

Effects : This routine initializes the instance variables.

void MakeItemBox ()

Effects : This routine constructs a VBox containing all the supertypes of the type object of which this supertype object is a component.

*** Public Operations

SuperObj (ObjectList* superlist, ButtonState* state, Type Object* typeobj, TypeStaticWindow* twindow, Command pop, Command shrink, Command follow)

Effects : Constructor of Supertype object of a type object. **state** is a button state of a type object. **typeobj** is a type object where this supertype object is belong to. **twindow** is TypeStaticWindow where the type object is displayed. **itemtype** is type of ListItem for this object. **superlist** is the list of supertypes of **typeobj**. **pop, shrink,** and **follow** are Commands for expanding, shrinking, and displaying full view of the type definition, respectively.

void Reconfig ()

Effects : This routine reconfigures the supertype object shape. It overwrites **Reconfig()** of **BaseObject** because the shape of supertype object is different from property and operation object.

2.6 PropObj

This section describes **PropObj** class which implements the property objects of a type object.

SuperClass : BaseObject
Instance Variables :

ObjectList* : proplist
List of properties of a type object.
ObjectList* : parmlist
List of parameters of a type object.
ListItem* : curitem
A property item currently selected.

*** Internal Operations

void Init (boolean flag)

Effects : This routines creates a string name for the title of the object. If **flag** is **True** it is Property. Otherwise, it is Parameter.

void MakeItemBox ()

Effects : This routine constructs a VBox containing all the properties of the type object of which this property object is a component.

void MakeParmBox ()

Effects : This routine constructs a VBox containing all the parameters of an operation of the type object of which the operation object is a component.

*** Public Operations

PropObj (ObjectList* proplist, ButtonState* state, TypeObject* typeobj, TypeStaticWindow* twindow, Command pop, Command shrink, Command follow, boolean flag)

Effects : Constructor of Property object of a type object. **state** is a button state of a type object. **typeobj** is a type object where this property object is belong to. **twindow** is TypeStaticWindow where the type object is displayed. **itemtype** is type of ListItem for this object. **proplist** is the list of properties of **typeobj**. **pop, shrink**, and **follow** are Commands for expanding, shrinking, and displaying full view of the type definition, respectively. **flag** determines the object is either of property or parameter. If **flag** is **True**, it is a Property. Otherwise, it is a Parameter object.

2.7 OperObj

This section describes **OperObj** class which implements the operation objects of a type object.

SuperClass : BaseObject
Instance Variables :

ObjectList* : operlist
List of operations of a type object.
ListItem* : curitem
An operation item currently selected.

*** Internal Operations

void Init ()

Effects : This routine initializes the instance variables.

void MakeItemBox ()

Effects : This routine constructs a VBox containing all the operations of the type object of which this operation object is a component.

*** Public Operations

OperObj (ObjectList* superlist, ButtonState* state, TypeObject* typeobj, TypeStaticWindow* twindow, Command pop, Command shrink, Command follow)

Effects : Constructor of Operation object of a type object. **state** is a button state of a type object. **typeobj** is a type object where this operation object is belong to. **twindow** is TypeStaticWindow where the type object is displayed. **itemtype** is type of ListItem for this object. **operlist** is the list of operations of **typeobj**. **pop, shrink**, and **follow** are Commands for expanding, shrinking, and displaying full view of the type definition, respectively.

3 Instance Browser

This section describes the classes used to implement**Instance Browser**. **InstStaticWindow** class provides data and operations to display and manipulate intsance objects so that the user can browse instances of ENCORE. **InstanceObject** is the base class for **InstObject, CollectObject**, and **SimpleObject. InstObject** class provides operations for constructing instance objects of single complex ENCORE type. **CollectObject** class provides data and operations for constructing the instances of Collection Type. **SimpleObject** is the class for displaying different view of both **InstObject** and **CollectObject.**

3.1 Class InstStaticWindow

InstStaticWindow class is a subclass of **GUIwindow** class, which ,in turn, is a subclass of **VBox** of InterViews. This class provides data and operations for implementing browser operations. Construction of actual instance objects is accomplished by uses of **InstanceObject, InstObject, CollectObject,** *and* **SimpleObject.**

SuperClass : GUIwindow
Instance Variables :

DisplayBox* : thebox
The box containing all the instance object displays of the **Instance Browser.**
BoxList* : boxlist
The list of top level boxes where the instance objects are placed.
OjectList* : typelist
The list of types whose instances are currently being displayed.
VBox * : dummyinnerbox
Dummy inner box containing no object. This is needed to hold the space when all the objects in the box are removed.
VBox * : dummyinnerbox
Dummy outer box containing no object. This is needed to hold the space when all the objects in the box are removed.
InstanceNode* : curinstnode
InstanceNode containing currently selected instance object.
ItemNode* : curitemnode
ItemNode containing currently selected item object.
void* : curitem
Currently selected item object.
boolean : instance
Flag for selection of **instance.**
boolean : collection
Flag for selection of **collection.**
boolean : nameobject
Flag for selection of **name object.**
boolean : simple
Flag for selection of **simple obje ct.**
ButtonState* : state
Buttonstate for InstStaticWindow.

*** Internal Operation

InstanceNode*
NewInstNode (void* instance)

Effects : This routine constructs an **InstanceNode** that stores all the information for the display of an instance object, **instance.** When the instance object is deleted, the corresponding node is also deleted.

virtual void
Activate ()

Effects : This routine activates the **InstStaticWindow.**

void
AddInstance (GUIinstance * instance)

Effects : This routine adds the new instance, **instance,** to the existing list of instances of the same type. It redraws the updated object, which is a CollectObject.

void
ChangeValue (GUIinstance* instance)

Effects : This routine changes the value of the selected instance. The instance containing changed item is redrawn with a new value.

void
ClearBoxes (InstanceNode* instnode)

Effects : This routine clears boxes that contains instance objects to be removed from the display because the instance object of **instnode** is simplified. The objects to be removed are chained from the **instnode**. The information for those objects are not deleted because they are temporarily removed from the screen.

void
DeleteBoxes (InstanceNode* instnode)

Effects : This routine deletes boxes that contains instance objects to be removed from the display. The objects to be removed are chained from the **instnode**. The information of deleted objects is also removed from the database of browser.

void
DeleteInstance (GUIinstance * instance)

Effects : This routine deletes **instance** from the display because corresponding model object was deleted from actual database. If the instance deleted has further chain from it, all those objects are also removed from the screen.

void
DisplayFront ()

Effects : This routine displays the full view of the selected instance object while simplifying all the fully displayed instance objects which were referred from same instance object as the selected one.

void
DisplayFull ()

Effects : This routine displays the full view of selected instance object while displaying all the fully displayed instance objects which were referred from same instance object as the selected one, if there are ones.

void
DisplayInstance (InstanceNode* instnode)

Effects : This routine redisplays the instance object contained in **instnode**. If there were more instance objects followed it, all those object also would be redisplayed with same shape before they were removed.

void
DisplayResult (GUIobject* guiobject)

Effects : This routine displays instance objects corresponding to **guiobject**, which was the result of query operation invoked in **Operation window**.

void
Expand (boolean flag)

Effects : This routine expands an object to its full view, which was previously simplified. If the flag is **True**, it will not shrink any full instance object of same group to simplified one. Otherwise, previous full view object is shrunken to simpler one.

void
FindBox (InstanceNode* instnode1, InstanceNode*& instnode2)

Effects : This routine assigns a box object that will contain the object in **instnode2** to the next box object following a box that contains **instnode1**. If the box is **dummy** box, it will create box and assign it to **instnode2**.

void
InnerBoxExist (InstanceNode* curnode, InstanceObject* new object)

Effects : This routine checks if there are other instance objects referred from the same instance object that this new type object was referred from. If so, it will place the new instance object at the bottom of the same **VBox** where those objects are placed. **newobject** is the new instance object to be displayed. **curnode** is the node containing all the information about current object from where the new object is referred.

void
OuterBoxExist (InstanceNode* curnode, InstanceObject* new object)

Effects : This routine inserts new instance object, **newobject** to the box where the objects referred from the object other than *curnode* are placed. Those object must be at the same outer level as with **newobject**. New box is created for **newobject** and subsequent object referred from **curnode** will be placed in the new box.

void
NoBoxObject (InstanceNode* curnode, BoxNode* outerboxnode, InstanceObject* newobject)

Effects : This routine creates both outer box and inner box because there is no objects displayed at the same level that this new instance object, **newobject** is to be displayed. A level is determined by number of objects displayed along horizontal box. For example, if an object A is at level 1, then any objects referred from A are displayed at the level 2.

void
Init ()

Effects : This routine initializes the instance variables for the **InstStaticWindow**.

void
InitMenus ()

Effects : This routine constructs the menus for the Instance browser. It inherits common menus such as **File** *and* **Edit** from *GUIwindow*. **Display** menus are created for the Instance Browser.

void
MakeItFront (InstanceNode* instnode)

Effects : This routine displays the full view of **instnode** and simplifies any fully displayed objects which are referred from same instance object.

void
MakeItSimple (InstanceNode* instnode)

Effects : This routine simplifies the object in **instnode** to its simple object. If there are any objects referred from it, they are temporarily removed from the display. Those object would be redisplayed if simplified object is expanded, later. An arrow will be displayed at the simplified object to indicate that there is further chain of objects.

void
RemoveInstance (InstanceNode* instnode, boolean flag)

Effects : This routine removes instance object in **instnode**. If the flag is **True**, it will delete also corresponding **instnode** because it is permanently deleted from browser database.

void
RemoveCollection (InstanceNode* collnode, boolean flag)

Effects : This routine removes collection object in **collnode**. If the flag is true, it will delete also corresponding **collnode** because it is permanently deleted from browser database.

void
RemoveSimple (InstanceNode* simplenode, boolean flag)

Effects : This routine removes simple object in **simplenode**. If the flag is true, it will delete also corresponding **simplenode** because it is permanently deleted from browser database.

InstanceNode*
SearchInstNode (ObjectNode* objnode)

Effects : This routine searches the entire list of instance nodes that are active in the browser to find the instance object in **objnode**. It returns an instance node corresponding to searched one if there is a such one. Otherwise, it returns nil.

void
TypeExist (ObjectNode* objnode)

Effects : This routine searches the list of types to find if there is a type whose instance is to be displayed. This routine is called when the user wants to display the list of instances belong to a type from either **TypeStaticWindow** or **TypeTarckingWindow**. If there is a such type, it means that corresponding instance(collection) is already displayed in the instance browser. Thus, it highlights that instance. Otherwise, it displays proper instance objects.

*** Protected Operation

virtual void
Handle (Event& event)

Effects : This routine invokes proper operation corresponding to the input either from the keyboard or the mouse.

virtual void
Reconfig ()

Effects : This routine reconfigures the instance browser window if there is any change in the configuration of the window.

virtual void
Resize ()

Effects : This routine is called when the instance browser window is resized.

virtual void
Copy ()

Effects : This routine copys the selected instance object to the buffer. Copied object is pasted into the **Clipboard Window.**

virtual void
Invoke ()

Effects : This routine invokes an operation on the selected instance object.

virtual void
ShowType ()

Effects : This routine displays the type object whose instance object is currently selected in the **Type Browser** window. If the Type browser is not currently opened, it will first open that window before it displays the type object.

*** Public Operation

InstStaticWindow (GUIappl* parent)

Effects : This routines creates an InstStaticWindow by calling **Init** *and* **InitMenus** routine. **parent** is the instance of **GUIwindow.**

DisplayBox*
GetBox ( )

Effects : This routine returns the box containing all instance objects being displayed in the browser.

void
ExamineInst (GUIobject * guiobject)

Effects : This routine will display the instance object corresponding to **guiobject. guiobject** is the result of a query operation.

void
FollowChain ()

Effects : This routine displays the contents of the selected item of **current instance** object in the full view. Since it only displays one full object among those object referred from **current instance**, it will simplify previous full object to simple one, if there is one. This routine calls **MakeItSimple**, which implements actual simplification and further removal of objects following simplified one.

void
NewChain ()

Effects : This routine starts new instance chain starting from the selected instance. The new chain will be displayed right below the last chain of the browser. The chain is created only when the selected instance is the last object of some previous chain .

void
NewInstList ()

Effects : This routine is similar to **NewChain** except it is called when an instance of ENCORE collection type is displayed. The type is selected from either **Type Browser** or *TypeTrackingWindow*.

void
NewInstance ()

Effects : This routine inserts and displays newly instance created by a creation operation. New instance is inserted into a collection instance where it is member of. The collection object is updated and redislayed.

void
RemoveObj ()

Effects : This routine removes instance object selected. Depending on the type of instance, it calls one of **RemoveInstance, RemoveCollection,** or **RemoveSimple** to actually remove it.

void
Simplify ()

Effects : This routine simplifies the selected instance object to simple object if it is not already simplified. This routine calls **MakeItSimple**, which is the routine to actually simplify the object.

void
UpdateInstance (GUIinstance* inst, ModificationType mtype)

Effects : This routine updates the display of instances currently on the screen, if there are any changes such as a deletion, an addition of an instance, or change of a value. **inst** is the instance object to be updated. ModificationType is one of **Add, Delete,** or **ChangeValue.**

3.2 Class InstanceObject

This section describes the **InstanceObject** class which implements instance object of browser. InstanceObject is a base class for **InstObject, CollectObject,** and **SimpleObject.** InstObject is the class for constructing instances of a single complex ENCORE type while CollectObject is the class for constructing instances of ENCORE collection type. SimpleObject is the class for representing different view of both InstObject and CollectObject when they are simplified.

SuperClass : VBox
Instance Variables :

InstStaticWindow* : pwindow
InstStaticWindow where this object is displayed.
VBox* : itembox
VBox* containing the list of items belong to this object.
ListItem* : nameitem
ListItem for the name of the object.
ClearPort* : viewport
ClearPort for the object.
ButtonState* : state
Button state of the object.

InstanceNode* : instnode
InstanceNode that contains this object.
ObjectList* : itemlist
List items of the object.
Command : namecommand
Command that are applied to the name of the object.
Command : itemcommand
Command that are applied to the other items beside the name of the object.
void* : object
Command object.
int : itemwidth
Width of the object.
int : itemheight
Height of the object.

*** Protected Operation

void
Reconfig ()

Effects : This routine reconfigures the instance object when there is any change in the shape of the object.

*** Public Operation

InstanceObject (ButtonState* state)

Effects : Constructor of InstanceObject. This routine initializes the instance variables. **state** is the button state for that instance.

~InstanceObject ()

Effects : Destructor of InstanceObject.

void
SetInstStat (InstStaticWindow* iwindow)

Effects : This routine sets **iwindow** to the InstStaticWindow where this object is to be displayed.

InstStaticWindow*
GetInstStat ()

Effects : This routine returns the InstStaticWindow where this object is displayed.

VBox*
GetItemBox ()

Effects : This routine returns a VBox containing the list of item belongs to this object.

Interactor*
ScrollFrame ()

Effects : This routine returns the vertical scroller for this object. The scrollbar is used to scrolls the contents of the instance, which is a list of values because size of the instance object is fixed while the list can be quite large.

ObjectList*
GetItemList ()

Effects : This routine returns the list of items belong to this instance object. The list of items is a list of values of properties of the type if it is **InstObject**. If it is **CollectObject**, a list of items is a list of **InstObject**.

void
GetCoord (Coord& xcoord , Coord& ycoord)

Effects : This routine returns relative x and y coordinates for the right top corner of the instance object in the browser. Coordinates are stored in **xcoord** *and* **ycoord**, respectively.

void
GetItemCoord (Coord& xcoord, Coord& ycoord)

Effects : This routine returns relative x and y coordinates for the left top corner of the instance object in the browser. Coordinates are stored in **xcoord** *and* **ycoord**, respectively.

void
MakeObject (GUIinstance* inst)

Effects : This routine make an instance(or collect object) by arranging the components of the object. The components are **name** of object and one of **SimpleObject**, **InstObject**, or **CollectObject**.

3.3 Class InstObject

This section describes **InstObject** class which is used for constructing instance objects of single complex ENCORE type.

SuperClass : InstanceObject
Instance Variables :

GUIinstance* : inst
GUIinstance object corresponding to this instance object.
ObjectList* : valuelist
List of property values of the type of this instance.
ItemNode* : curitem
Currently selected ItemNode.

*** Internal Operation

void
Init ()

Effects : This routine initializes the instance variables.

void
MakeItemBox ()

Effects : This routine forms a box containing all the list of values for the instance. The values corresponds to the values of the properties for the type of the instance.

*** Public Operation

InstObject (GUIinstance* guiinst, ButtonState* state, Command namecom, Command itemcom , void* what) : (state)

Effects : Constructor of InstObject. **guiinst** is a GUIinstance object corresponding to this instance object and **state** is a button state of the object. **namecom** and **itemcom** are commands that applied to the name object and value objects, respectively. **what** is a Command object.

void
SetCurItem (ItemNode*, boolean)

Effects : This routine sets **itemnode** to the current node that contains the currently selected item object. **titleflag** is **True** if selected item is name object, the name of the instance object.

ItemNode*
GetCurItem ()

Effects : This routine returns itemnode whose item object is currently selected.

3.4 Class CollectObject

This section describes **CollectObject** that are used for implementation of instances of ENCORE collection type. CollectObject is a collection of **InstObject**.

SuperClass : InstanceObject
Instance Variables :

Type* : type
Type of the collection object.
GUIcollection* : coll
GUIcollection object corresponding to this collection.
ObjectList* : instlist
List of instances that are values of the type of this collection.
ItemNode* : curitem
Currently selected ItemNode.
boolean : collection
Flag indicating the reference.

*** Internal Operation

void
Init ()

Effects : This routine initializes the instance variables.

void
MakeTypeBox ()

Effects : This routine constructs a collection box when this object is referred from **Type Browser**. The box contains the list of instances of a selected type.

void
MakeCollectionBox ()

Effects : This routine constructs a collection box when this object is referred from some other instance objects in the **Instance Browser**. The box contains the list of instance of the type whose instance is selected.

void
MakeObject (GUIobject* coll)

Effects : This routine makes a collection object for **coll** by arranging the components. The components are name, a list of instance items and attached vertical scroller.

*** Public Operation

CollectObject (GUIobject* guiobj, ButtonState* state, boolean flag, Command namecom, Command itemcom, void* what) : (state)

Effects : Constructor of CollectObject. **guiobj** is a GUIobject corresponding to this object. **state** is a button state of this object. **namecom** and **itemcom** are commands that are applied to name object and other items, respectively. **what** is a Command object. **flag** is **True** if the collection object is referred from the Type Browser. Otherwise, it is **false.**

void
SetCurItem (ItemNode* itemnode , boolean titleflag)

Effects : This routine sets **itemnode** to the current node that contains the currently selected item object. **titleflag** is **True** if selected item is name object, the name of the collection object.

ItemNode*
GetCurItem ()

Effects : This routine returns ItemNode whose item object is the current selection.

void
Reconfig ()

Effects : This routine reconfigures the collection object when there is any change in the shape of it.

3.5 Class SimpleObject

This section describes **SimpleObject** class which implements simplified version of both **InstObject** and **CollectObject**. The objects in the browser are simplified to save the space when they are not current interest of the users. Objects of this class contains only one component, which is a name of an instance.

SuperClass : InstanceObject
Instance Variables :

GUIobject* : guiobj
GUIobject corresponding to this simple object.
ItemNode* : curitem
ItemNode whose item object is currently selected.
boolean : selected
Flag indicating the selection of this object.

*** Protected Operation

virtual void
Reconfig ()

Effects : This routine reconfigures the shape of the simple object. The object is redrawn using this new configuration.

*** Public Operation

SimpleObject (GUIobject* guiobj, ButtonState* state, InstanceType insttype, Command command, void* object, boolean arrowflag)

Effects : This routine constructs a simple object. **guiobj** is GUIobject corresponding to this simple object. **state** is the button state of the object. **insttype** is type of instance to be simplified. It is one of **Instance**, or **Collection**. **command** is the command that applies to this object and **object** is the Command object. **arrowflag** indicates whether this object needs an arrow icon or not.

void
SetCurItem (ItemNode* itemnode)

Effects : This routine sets **itemnode** to the current itemnode whose item is selected.

ItemNode*
GetItemNode ()

Effects : This routine returns the current itemnode whose item is selected.

3.6 Class ListItem

This section describes **ListItem** class, which is a subclass of **TextButton** class of Interviews. ListItem class provides data and operations for constructing unit component of a type or an instance object. The unit element is composed of an icon, and string describing the model object which it represents. It can be name of an object, or property, or operation, etc. Since it is a subclass of **TextButton**, the item can be selected and highlighted.

SuperClass : TextButton
Instance Variables :

IconType : icontype
Icon type of the item. Icon types are **Simple**, **Complex**, and

Collection.

ItemType : itemtype
Item type of the item. Item type can be one of **Property, SuperType, Operation, Instance, Collection,** *Name.*

Type* : type
ENCORE type of the object the item represents.

BaseObject* : baseobj
Baseobject where the item belongs to.

GUIobject* : guiobj
GUIobject corresponding to the item.

ObjectNode* : objnode
ObjectNode containing the item.

GUIbaseNode* : basenode
BaseNode containing a BaseObject where the item belongs to.

void* : object
Command object.

Command : command
Command that can be applied to the item.

HLstate : highlight
Highlight state of the item.

int : curGroupId
Group Id whose element is currently being fully highlighted.

int : myGroupId
Id of the group where the item belongs to.

int : itemwidth
Width of the item.

int : itemheight
Height of the item.

char* : title
String name for the face of the item.

boolean : arrow
Flag for arrowicon's existence.

Bitmap* : arrowicon
Arrow icon of the item.

Bitmap* : icons
Type icon of the item.

*** Protected Operation

virtual void
Reconfig ()

Effects : This routine reconfigures the shape of the item when there is any changes in the configuration. Consequently, the item will be redrawn using new configuration.

virtual void
Refresh ()

Effects : It refreshes the display of the item when there is any change in the item. For example, if the item is selected, it will display the item in reverse video. It also reshapes the item when shape of the item is updated.

virtual void
Redraw (Coord x1 , Coord y1, Coord x2, Coord y2)

Effects : This routine redraws the item by calling **Refresh.** (x1, y1) is lower left corner and (x2, y2) is upper right corner of the canvas.

virtual void
Press ()

Effects : This routine takes appropriate action for the item being pressed. It sets the associated state to the button's value, which will trigger all related items to update their display.

*** Public Operation

ListItem (char* name, ButtonState* state, GUIobject* guiobj, IconType icontype , ObjectNode* objnode, ItemType item type, int gid)

Effects : Constructor of ListItem. **name** is string for the face. **state** is the buttonstate. **guiobj** is GUIobject corresponding to this item. **icontype** is type of icon and **itemtype** is a type of item for this item. **objnode** is the ObjectNode containing this item. **gid** is the group id for this item. Its default is 0.

ListItem (char* name, ButtonState* state, Type* type, Icon Type icontype, GUIbaseNode* basenode, ItemType item type BaseObject* baseobj, Command command, void* object, boolean arrowflag, int gid)

Effects : Alternative constructor for ListItem. **type** is the ENCORE type for the object represented by this item. **basenode** is the BaseNode containing the BaseObject where the item belongs to. **baseobj** is the object where the item belongs to. **command** is the command that can be applied to this item and **object** is the command object. **arrowflag** indicates whether the item has an arrow or not.

virtual
ListItem ()

Effects : Destructor of ListItem.

virtual void
Update ()

Effects : This routine will updates the highlight state of the item. The selected item will be fully highlighted while other items currently being highlighted will be updated depending on the group of the selected item. Any item in the same group with the selected item will be dehighlighted while items in the different group will be half highlighted if they were fully highlighted. All the listitem that belong to same object have same group id.

void
Choose ()

Effects : This routine will fully highlight the item.

void
HalfChoose ()

Effects : This routine will halfhighlight the item.

void
UnChoose ()

Effects : This routine dehighlight the item.

void
SetIcon ()

Effects : This routine creates a bitmap for the icon of the item. It can be one of **Simple, Complex, Collection, Title,** or **Object.**

TypesNode *
GetTypeNode ()

Effects : This routine returns the typenode that contains the type object of which this item is member.

void
GetCoord (Coord& xcoord, Coord& ycoord)

Effects : This routine returns x and y coordinates for top right corner of a box containing definitions of either type or instance object where this item belongs to.

void
UpdateMenus ()

Effects : This routine updates the display of the menus corresponding to the list item selected in the browser. That is, if an item of supertype is selected, the menus for supertypes will be changed to normal from dimmed state while all other menus that are not applicable to the selected item will be disabled.

3.7 Class DisplayBox

This section describes **DisplayBox** class implementing composition of instance objects in the Instance Browser. DisplayBox is a subclass of **HBox** of InterViews which arranges its components horizontally. It provides a drawing space for instance objects of Instance Browser.

SuperClass : HBox
Instance Variables :

InstStaticWindow* : instwindow
InstStaticWindow where the box is in.
BoxList* : boxlist
List of boxes contained in this box.
boolean : flag
Edge draw flag.

*** Internal Operation

void ItemRefer (InstanceNode* instnode)

Effects : This routine finds location of objects that are updated by some operation. It iterates every item (value objects) of currently selected object and checks if there are any changes among those object referenced from those items. If there are any changes such as removal, shrieked, it will find new location for those object. This is done recursively until it reaches a level where objects are not affected by current operation.

*** Public Operation

DisplayBox ()

Effects : Constructor of DisplayBox. flag is set to false to indicate that it does not need to draw edges for its components.

void FindShape ()

Effects : This routine finds space where new instance object is displayed. It reconfigures the DisplayBox after allocating the space.

void DrawEdge ()

Effects : This routine draws an edge to left top of an object,B, from an object,A, containing the item that is related to B. The edge goes from top right corner of A to top left corner of B.

3.8 Class TypeBox

This section describes **TypeBox** class implementing composition of type objects in the Type Browser. TypeBox is the outermost box object which provides the drawing space for type objects.

SuperClass : HBox
Instance Variables :

TypeStaticWindow* : typewindow
TypeStaticWindow where the box is in.
BoxList* : boxlist
List of boxes contained in this box.
boolean : flag
Edge draw flag.

*** Internal Operation

void SuperTypeRefer (TypesNode* typenode)

Effects : This routine checks if there is any change among the type objects referred from the supertype objects of current type object in typenode. This routine is called by **DrawEdge**.

void PropRefer (TypesNode* typenode)

Effects : This routine checks if there is any change among the type objects referred from the property objects of current type object in typenode. This routine is called by **DrawEdge**.

void OperRefer (TypesNode* typenode)

Effects : This routine checks if there is any change among the type objects referred from the operation objects of current type object in typenode. This routine is called by **DrawEdge**.

void ParmRefer (TypesNode* typenode)

Effects : This routine checks if there is any change among the type objects referred from the parameter objects of current type object in typenode. This routine is called by **DrawEdge**.

*** Public Operation

TypeBox ()

Effects : Constructor of TypeBox. flag is set to false to indicate that it does not need to draw edges for its components.

void FindShape ()

Effects : This routine finds space where new instance object is displayed. It reconfigures the DisplayBox after allocating the space.

void DrawEdge ()

Effects : This routine finds location of objects that are updated by some operation. It iterates every item (value objects) of currently selected object and checks if there are any changes among those object referenced from those items. If there are any changes such as removal, shrunk, it will find new location for those object. Then, it draws an edge from an object A to another object B if they are directly related by some refer–referred relation. This is done recursively until it reaches a level where objects are not affected by current operation.