# Browsing and Querying in Object-Oriented Databases

Juliano Lopes de Oliveira
Ricardo de Oliveira Anido *
Departamento de Ciência da Computação
UNICAMP - BRAZIL

**Abstract**

We present a new interface for Object-Oriented Database Management Systems (OODBMSs). The *GOODIES*[1] system combines and expands the functions of many existing interface systems, introducing some new concepts for improved browsing in an OODBMS. The implementation of GOODIES proposes a new approach to database interfaces development: instead of being strongly dependent of the underlying DBMS, GOODIES is based on the main features of the object-oriented data model. The system design is based on an internal model and on an external model. The internal model defines the relationships that bind the interface to the DBMS. The external model determines the possible interaction between the user and the interface system. This paper describes the concepts of the design of GOODIES.

**Keywords:** object-oriented databases, graphical interfaces, direct manipulation paradigm, browsers, query tools.

## 1 Introduction

Offering database users a suitable interface is an old research issue, and much work has being done towards that objective. Database Management Systems (DBMSs) are powerful software tools, with a large and complex set of functions. The main purpose of interface systems for DBMSs is to improve access to those functions for the whole database (DB) user community. That, however, is not an easy task, since different kinds of users (application programmers, database administrators and end-users) expect different, and sometimes conflicting, functions. As graphical workstations become more popular, there is a strong trend to substitute the traditional DB programming languages by the graphical interfaces, which are more suitable for the interaction between the user and the DBMS [MvD91, Shn87, PH91].

In this paper we introduce GOODIES, a new system for browsing and querying in Object-Oriented Database Systems (OODBMSs). This new system is a multiple window graphical interface using the direct manipulation paradigm, and supporting multi-media objects. The system combines, in a single tool, the main functions for database browsing at both schema and data levels. Another important feature of GOODIES is to be independent from a specific OODBMS.

### 1.1 Graphical Interfaces for DBMSs

[GGKZ85] presents the ISIS system, a graphical interface for the semantic data model. This system permits both schema and data manipulation. The relationships among the objects are displayed as lines that bind the schema classes. Data is displayed in separate windows, one window for each object's class. Another interface for the semantic data model, the SNAP system, is presented in [BH86]. SNAP provides facilities for schema manipulation and query formulation. The schema is presented as a very complex graph, where different geometric figures are used to represent different relationships.

A graphical interface for the entity-relationship data model is presented in [RC88]. The system permits navigation and update of both schema and data. The interface automatically creates presentations for the entities defined in the schema, and the user can modify these presentations. During navigation, the system operates in two modes: *browse*, where the user cannot modify the information, and *edit*, where update operations are allowed.

The PICASSO system [KKS88] introduces a graphical query language for DBs. The query formulation is based on a mouse with three buttons. The left button is used to select attributes; the middle button is used to build predicates; and the right button is used to choose options from the query processing menu. Thus, PICASSO allows a graphical definition of queries, and the queries defined in the system are very similar to the well-known SQL's query blocks. An auxiliary tool allows both navigation through the results of executed queries, and formulation of complex queries using the results of previous queries.

KIVIEW [MDT89] is an object-oriented system that improves the access of non-expert users to a DB. It allows navigation on both schema and data levels. KIVIEW is a powerful browsing tool because the user can save information during the navigation process, and the saved information can be used as a starting point for other navigations. KIVIEW also allows the simultaneous navigation in objects of different classes, through *synchronized browsing operations*.

The LOOKS system is a graphical presentation generator for the OODBMS O2. [Alt90a] describes the primitives provided in LOOKS to manipulate the presentations using a

*Research partially financed by grant CNPq-Brazil 500869/91-0

[1] GOODIES is an acronym for *Graphical Object Oriented Database Interface with Extended Synchronism*.

programming language. The LOOKS architecture is presented in the second part of [Mam91]. Besides the LOOK presentation generator, the O2 system has an object-oriented programming environment called OOPE [Alt90b]. Among other functions, OOPE allows: creation, navigation and edition of classes and methods, visualization and edition of the class hierarchy and *ad hoc* query execution. The association of OOPE with LOOKS gives access to the whole set of functions of the O2 system, and they are considered a complete OODBMS interface system [BMP+92].

Other existing OODBMS interface systems can be cited, although they are not as powerful as the O2 DBMS interface system. ODEVIEW [AGS90], the interface system for the ODE OODBMS, allows schema and data manipulation. A special function in ODEVIEW permits the simultaneous navigation among objects of different classes (synchronized browsing). In [Alm91] it is presented the GS-Designer system, an interface tool that allows the graphical interactive definition of classes and relationships for the OODBMS GemStone.

### 1.2 A New Interface for OODBMSs

Unlike relational databases, which share exactly the same data model, OODBMSs are not based on a common formal model. Indeed, the object-oriented (OO) data model is composed by a set of properties and functions that database researchers consider essential for a DBMS to be accepted as an object-oriented system. Recently, many papers proposed basic features that should be present in an OODBMS ([ABD+89], [Com90], [Jac91], [Cat91] and [BM91]). The following characteristics represent the common points in these propositions:

1. To have the basic features of a *complete DBMS*;
2. To support *complex objects* and *object identity*;
3. To provide *encapsulation*;
4. To support the *class* concept, and to permit *inheritance* and class hierarchies;
5. To allow *overloading* and *late binding* of methods;
6. To be *extensible* and *computationally complete*.

Therefore, OODBMSs implement similar features, but they do not follow an specific set of rigid rules. Due to this diversity of features in OODBMs, interface systems for OODBMSs have an *ad hoc* design, according to the specific implementation used in the OODBMS for the fundamentals of the OO data model.

The GOODIES system introduces a new approach to the construction of interfaces for OODBMSs. Discarding the idea of a strong relationship between the OODBMS implementation and the process of interface development, GOODIES's design was directed by the essential features of the OO data model, identified above, independently of a specific implementation of these features [Oli93].

This new approach to OODBMSs interface development presents some advantages in comparison to the previous approach. First, it permits the validation of the basic features that define the OO data model. A second advantage is that it permits to verify whether a given DBMS provides these features, that is, the new approach can be used to verify if the DBMS is object-oriented. Finally, the new approach facilitates the adaptation of the interface system to a DBMS that implements, in any way, the basic object-oriented features.

At the present stage, the GOODIES system implementation only provides reading access to the information stored in the DBs. Thus, the system cannot be considered a complete DBMS interface system. However, the GOODIES system design was conceived with the objective of being extensible. So, the information update capability can be incorporated to the system, without changing its external model (user's view of the system), through a reduced number of modifications on the internal model of the system (the way the system dialogs with the underlying DBMS).

The following sections describe the design of this new interface system. Section 2 presents some concepts of the internal model of GOODIES. In section 3 we describe the way the information is represented in the system. Section 4 shows the interaction mechanism between the user and the interface. In section 5 we explain the behavior of the browse and query operations. Section 6 introduces some functions that improves the system utilization. The last section comments the system implementation and relates it to previous work.

## 2 GOODIES Concepts

A OODBMS can control many DBs, and GOODIES defines a DB as an schema and a set of data. The set of data represents the DB objects, while the schema is represented by the inheritance and composition graphs, and by a set of methods.

### 2.1 Inheritance Graph

The *inheritance graph* is a directed acyclic graph whose nodes represents all the schema classes. Each node is labeled with a class name, and identical class names are not allowed within the same schema. The nodes are also associated to a list of methods defined for the class they represent.

There are two kinds of edges in the inheritance graph: *specialization* edges and *generalization* edges. The *generalization* edges are directed from the subclasses to the superclasses, and they represent a generalization relationship between a subclass (origin of the edge) and a superclass (destine of the edge). In the same way, *specialization* edges are directed from the superclass to the subclass, representing a specialization relationship. It is easy to see that an edge linking two nodes of the graph has always a complementary edge of different kind and opposite direction.

GOODIES internal model supports the multiple inheritance concept, since a node in the inheritance graph may have an arbitrary number of both kinds of edges. Following the basic features of the OO model, a specialized class inherits attributes and methods from its superclasses.

### 2.2 Composition Graph

The *composition graph* is a directed graph that may contain cycles. There are three kinds of nodes in this graph: *class nodes*, *constructor nodes* and *attribute nodes*. A class node is labeled with the name of the schema class it represents. For each schema class there is one, and only one, class node in the composition graph.

Constructor and attribute nodes represent the class composition, and they are subordinated to class nodes. An attribute node is labeled with the type of the attribute it represents. It is possible to have duplicated labels for attribute nodes. The possible values for attribute labels are: Simple, Text, Image and Sound. Sub-objects are represented in the composition graph as references to class nodes. Constructor nodes are labeled with the type of constructor in the GOODIES internal model. There just two types of constructor in GOODIES: Tuple, used for collections of heterogeneous elements, and List, used for collections of elements that belong to the same type (for instance, sets and list of elements).

An edge in the composition graph connects a compositor node to a component node (it is directed from the first to the former), and it is labeled with the name of the attribute defined in the class type. Edges connecting nodes that are subordinated to the same class node can not have identical labels.

A class node origins an edge for each attribute defined in the class type. Similarly, a class node has an incident edge for each attribute whose type is the represented class. A subordinated node may reference its class node, and this is the way GOODIES allows an object to have sub-objects of its own class.

A constructor node of type *List* has one, and only one, incident edge, since each list node is subordinated to a unique class node. This constructor origins also one, and only one, edge which points to a node that defines the type of the elements of the list. As we have already said, the GOODIES list constructor is used for both list and set constructor of the OO model.

For the same reason described for the list constructor, a constructor node of type *Tuple* has only one incident edge. However, a tuple node may origin an arbitrary number of edges (one edge for each element of the tuple). Each of these edges points to a node that defines the type of the element.

The attribute nodes represents the atomic attributes of GOODIES, and they can not give origin to edges. These nodes always have one incident edge, labeled with the name of the attribute whose type is defined by the attribute node.

### 2.3 Primitive Operations

GOODIES interacts with a OODBMS through *primitive operations*. The semantics of these operations are defined by GOODIES, while their implementation is dependent of the OODBMS. There is a module in GOODIES that is responsible for the primitive operations, and it is the only module that is dependent of the underlying OODBMS. It is important to understand that only the implementation of the operations is variable; their semantics are specified in the GOODIES internal model, and are independent from a specific DBMS.

The primitive operations were designed as a minimum set of functions that should be provided by a OODBMS in order to grant the user access and control of the DBs. In fact, the primitive operations may be seen as textual queries about the schemas and about objects in the DBs. The primitive operations defined by GOODIES are:

**Get-Schema:** the objective of this operation is to obtain from the OODBMS the list of classes defined in a given DB schema. To reach this objective, the operation receives a DB name as a parameter. The answer provided by the OODBMS is interpreted and stored in internal data structures of GOODIES. The result of the operation is presented to the user through the DB window (figure 2).

**Get-Class:** GOODIES uses this operation to get the complete description of a given schema class. The class and the schema are received as a parameter of the operation. The class description is composed by the class type (or composition) definition and by the lists of superclasses, subclasses, methods and objects that belong to the class. The result of this operation is used to build the inheritance and composition graphs, and these informations are displayed in the class window (figure 3).

**Get-Object:** This operation has three parameters, that are the schema, the class and the object identifier (oid) of the desired object. A query is formulated to the OODBMS, which should answer with the values of the attributes of the given object. Using the inheritance and composition graphs, GOODIES parses this answer and presents it to the user through the object window (figure 4).

It is obvious that both the command and the format of the answer are dependent from the underlying OODBMS. Thus, to adapt GOODIES to a particular OODBMS, it is necessary to: a) determine the syntax of the OODBMS commands that corresponds to the semantics defined by GOODIES; b) adapt the answer interpretation function of GOODIES to the format used in the OODBMS. This adaptation process is fully described in [Oli93].

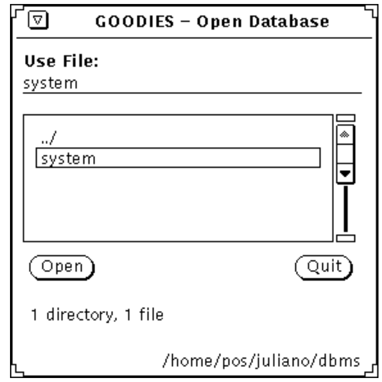


Figure 1: Directory Window

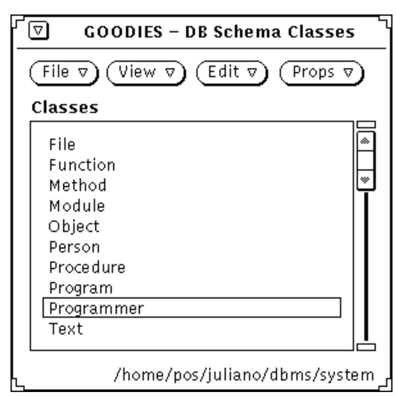


Figure 2: DB Window

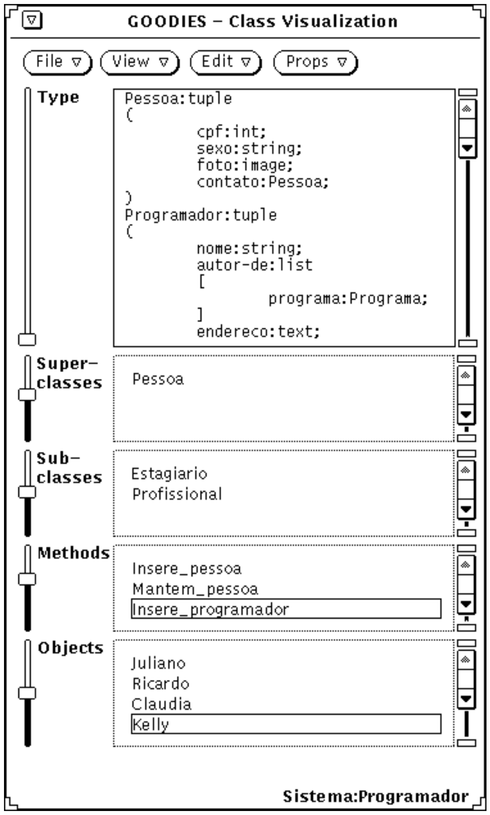


Figure 3: Class Window

## 3 Information Visualization

In GOODIES, all kinds of information are displayed through windows. Windows are composed by three parts: header, body and footer. The window title (that is, the identification of the kind of information displayed in the window) appears in the header. The window body contains the controls and the representations of the information associated to the window. The window footer is split in two parts: left and right. In the right part it is presented the name or identification of the DB component that is represented in the window's body. The left footer is reserved for system messages related to either the presented data or to the operations performed on the window.

The system has four types of base windows, where the information about schema and data (objects) are displayed. There is also a set of auxiliary windows, which allows the user access to the complete system functionality. GOODIES allows an arbitrary number of windows to be displayed simultaneously.

### 3.1 Schema Visualization

Three base windows contain information about schemas: the *directory window*, which provides browsing facilities at DB level; the *DB window*, which presents the list of classes that are defined in a given DB schema; and the *class window*, which presents the items that define a given schema class.

The directory window provides access to the existing DBs. This window allows navigation on the file system in order to select DBs. The user can visualize different DBs at the same time, since each DB selection in the directory window opens the DB window corresponding to the selected DB. Existing DBs in a directory are visualized through a list in the directory window. This list contains also the subdirectories of the visualized directory. Figure 1 shows a directory window, and figure 2 presents the DB window that contains the classes defined in the schema of the *system* DB.

The third base window for schema visualization is the class window. This window presents the definition of a class in a DB schema, and it is composed by the following items:

- **Type**: a textual description of the class type definition, that is, the composition of the instances (objects) of the class;
- **Superclasses**: a list of superclasses from which the described class inherits attributes and methods;
- **Subclasses**: a list of subclasses that inherit the attributes and methods defined for the described class;
- **Methods**: a list of methods associated to the described class;
- **Objects**: a list of object instances that belong to the described class, that is, the class extension. If the underlying DBMS supports named objects, the objects names appear in the list. Otherwise, the list will contain a sequence of items [*object 1*, *object 2*, ..., *object n*]. Each of these items represents a single object in the class extension.

Figure 3 shows a class window that displays the class *Program* of the DB presented in figure 2. The sliders on the left of the list items allow the resizing of the representation of a given list item with respect to the other items. The system automatically changes the size of the items in such a way that the complete set of items continue to be displayed in the available space. This mechanism is useful to show more information on important items.

### 3.2 Data Visualization

The three base windows described in the previous section (directory window, DB window and class window) are used to visualize and to navigate on the schema definitions of the different DBs controlled by an OODBMS. The fourth base window permits the execution of these operations on data, i.e., on the objects stored in the DBs.

The *object window* contains the values of the attributes that compose an object instance, according to the class composition description presented in the class window. Figure 4 shows an object of the class *Program*, presented in figure 3.

The objects attributes are divided, according to their representation in the system, in the following groups:

1. Simple Attributes: these attributes are those which can be displayed as character strings containing at most 128 characters, and that are *atomic*, that is, they are not composed by other elements. Numbers (real, integer), boolean values and character strings [2] are examples of simple attributes. These attributes are represented directly in the object window. The attributes *objective* and *identification* of figure 4 are simple attributes.

[2] character strings are not considered to be composed by elements of type character because, in this case, the individual characters do not have their own semantic meaning

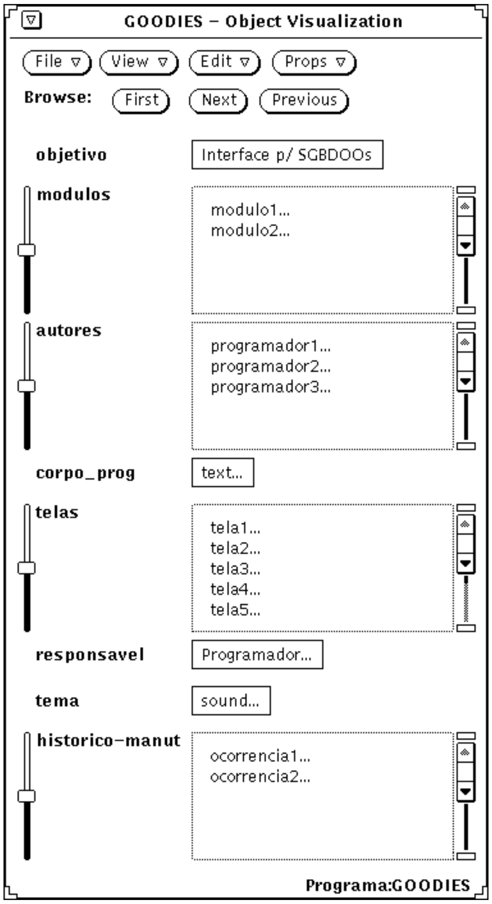


Figure 4: Object Window

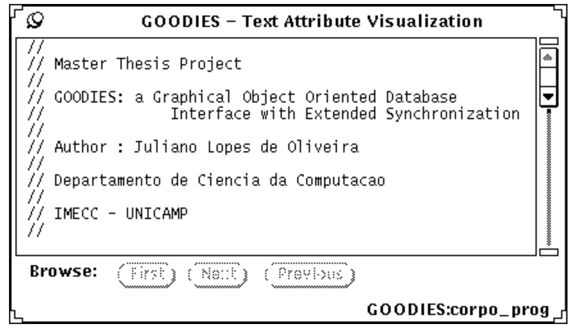


Figure 5: Text Window

2. Textual Attributes: in this group are the atomic attributes, as defined above, which cannot be represented with less than 129 characters. These attributes are displayed in auxiliary *text windows*, associated to the object window that contains the textual attribute. Figure 5 shows the representation of the textual attribute *program_body*, of the object presented in figure 4.

3. *Images*: an image is a sequence of bytes that defines the graphical representation of a picture. Images are presented in auxiliary graphical windows, associated to the object window that contains the image attribute. Figure 6 shows the image window that corresponds to the first element of the *windows* list of the object presented in figure 4.

4. *Sounds*: sound attributes are applied to audio recordings, whose representation is realized by reproducing the sound stored in the attribute. The sound and image attributes provide facilities for storing and manipulating of *multi-media* objects, which are supported by the majority of existing object-oriented systems.

5. Lists: collections of elements (obtained through the constructors *set* and *list*, for instance) that belong to the same type are represented by a list attribute. The elements of a list may be either simple or complex. Simple attributes are displayed directly in the object window as list items. If the elements of the list are not simple attributes, the items of the list presented in the object window work as references to the attributes that must be presented in auxiliary windows. It is important to note that the list attribute of GOODIES do not correspond directly to the type constructor *list* of the OO model, since that attribute is also used to represent sets of elements.

6. *Tuples*: tuple attributes represent the aggregation of elements of heterogeneous types (in general these attributes are defined through the constructor *tuple*). Thus, tuples demand the creation of an auxiliary window in order to display its contents, since each tuple element may belong to any of the defined attribute types.

7. Sub-objects: these attributes are used to represent the concept of *complex object*. According to this concept, an object can be composed by an arbitrary set of other objects. The sub-objects are displayed in object windows associated to the base object window. There is no difference between the construction and presentation of sub-object windows and the construction and presentation of object windows, except that the sub-object window is associated to the base object window, whereas the base object window is associated to the object's class window. This subtle difference is the base of the *synchronized browsing* capability described later in this text.

The auxiliary windows associated to the object window follows the same scheme for attribute representation used in the object window. Thus, it is possible to represent an arbitrary number of nested objects and values, and this satisfies the directives for objects construction in the OO data model [ABD+89]. The attributes that must be visualized in different windows are easily identified, because their reference names are ended with ellipses ("..."), as shown in figure 4.

## 4 Interaction with the user

The direct manipulation paradigm [Shn83] was adopted as the main mechanism for interaction with the user. This mechanism simplifies the input actions required from the user in order to execute an operation, and reduces both the amount of input errors and the user typing effort.

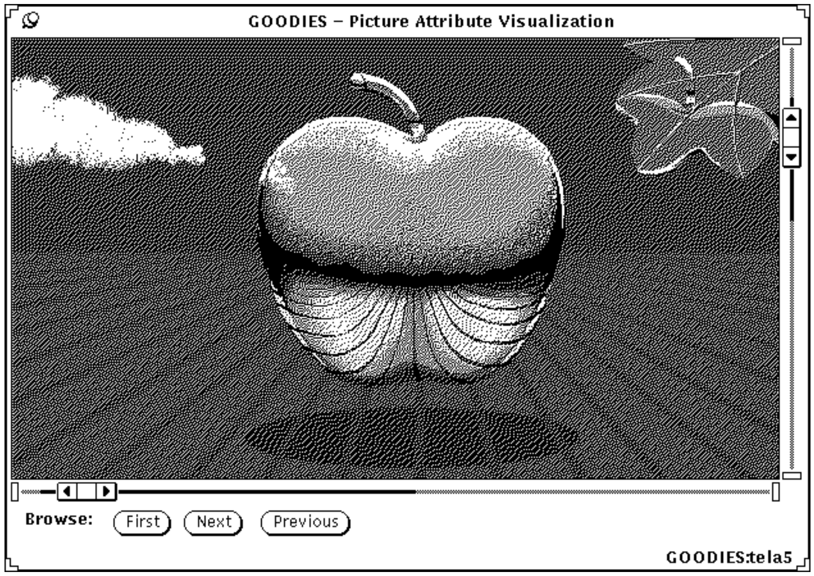


Figure 6: Image Window

The coherence between actions and results was a major guideline of the system design, as it guarantees that the final user will have a quick understanding of the interface functionality.

Besides assuring coherence, the user interaction mechanisms of GOODIES also provide flexibility for the user to define the environment where he is going to work. GOODIES allows the user to set up his workspace, through facilities to resize, reposition, open, close, create and destroy windows. The system neither limits the number of opened windows (in fact this number is limited by the *Window Manager* and by the available memory in the equipment), nor imposes any kind of restriction about size or positioning of the windows.

## 5 Mechanisms for Browsing and Querying

Up to this point we presented the available windows in the GOODIES system. The next sections describe how these windows are used to visualize different aspects of schema and data contained in a OODBMS.

### 5.1 Schema Level Navigation

A working session in GOODIES is initiated with the directory window, which allows the user to choose the desired DBs. The selection of a DB causes the presentation of a DB window, containing the list of classes defined for the selected DB.

Once obtained a DB window, the user can select the schema classes that he wants to visualize from the DB window classes list. By selecting classes in this list the user obtains the corresponding class windows, which contain the complete description of each schema class (section 3.1 presents and explains the contents of the class window).

In a similar way, starting from the class window, the user can proceed browsing the schema either selecting classes from the subclasses and superclasses lists, or selecting methods from the class methods list. It is also possible to start browsing over the class objects, through the selection of instances in the class objects list.

The selection of superclasses or subclasses in the class window represents exactly the same operation of selection classes in the DB window. These operations cause the creation and presentation of the class windows for the selected classes.

The selection of a method from the class methods list triggers the process of creation and presentation of an auxiliary window, the method description window. This window contains the textual description of the selected method, and each method selection causes the creation of a new method window. Figure 7 shows the presentation of a method of the class exhibited in figure 3.

### 5.2 Data Level Navigation

The data level navigation starts with the selection of an object from the objects list of a class window. This operation causes the presentation of an object window for the selected object, and each new selection in that list causes the creation of a new object window. Thus, the user can work with many instances of the same class simultaneously.

*Sequencing operations* are available to provide access to different objects through a single object window. These operations are activated by the *next*, *previous* and *first* buttons of the object window. The *next* button updates the contents of the object window with the value of the next object in the class objects list.

The *previous* button has an analog effect, except that instead of using the next element, it uses the previous element in the class objects list. The *first* button causes the presentation of the first element of the class objects list, no matter what object is currently being visualized in the object window.

It is worth noting that the sequencing operations *next* and *previous* see the class objects list as a circular list, in such a way that the activation of *next* on the last element of this list causes the presentation of the first element, and the activation of *previous* on the list's first element exhibits the last element of the list on the object window.

## 5.3 Query Facilities

Up to this point we described the basic mechanisms for navigation in GOODIES. These mechanisms are also present in many other existing database interface systems. This section introduces the additional capabilities that improve the browsing power of GOODIES, and which can be regarded as a simplified querying process.

It is important to distinguish at this point the adopted terminology: *browsing* (or *navigation*) is the process of sequential visualization of information of a specific type; *querying* is the process of selecting and restricting information, in such a way that only the explicitly demanded information is retrieved from the DB and presented to the user.

### 5.3.1 Predicates

The first query facility available in GOODIES is the formulation of *predicates*. The *Props* menu in the object window has a *"predicate..."* option that creates an auxiliary window associated to the object window. This auxiliary window is the predicate window, where the user can define predicates that are applied to the object presented in the associated object window. A predicate is composed by three elements:

**Attribute:** An attribute of the object displayed in the object window for which the predicate window was created;

**Operator:** Either a comparison operator $(=, <, >, \leq, \geq, \neq)$ or a set operator $(\supset, \subset)$;

**Referential:** Either a value or an attribute of an object presented in the user workspace. If the referential is an attribute, its type must be compatible with the type of the first element of the predicate.

A predicate can also be composed by the association of other predicates, through logical connectors (*And, Or*) and the logical negation operator (*Not*). Parentheses can be used to specify a resolution order for the composed predicates.

Once the predicate is defined by the user, the semantic of the sequencing operations for the associated object window is modified. The activation of *next* will not find the next

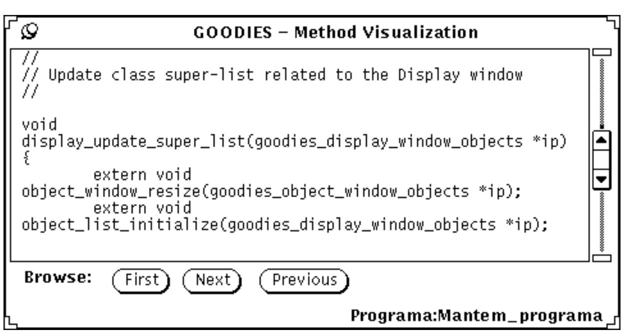


Figure 7: Method Window

element of the class objects list, but the next element of this list that satisfies the defined predicate. The same behavior is adopted by the *previous* operation, that searchs the list in the reverse order, and by the *first* operation, which finds the first element, starting from the beginning of the list, that satisfies the defined predicate.

### 5.3.2 Syncronization

Another query facility provided by GOODIES is the *synchronization* of object windows. Since an object window can have references to other objects (sub-objects), the act of opening an object window through these references creates a synchronization link between the complex object window and the sub-object window. Each reference to a sub-object can have many associated windows, forming a *synchronization tree*. The synchronization mechanism guarantees that any sequencing operation applied on an object window is reflected in the whole sub-tree whose root is the object window on which the sequencing operation was performed.

A synchronization link creates a relationship of hierarchy between two object representations. However, a synchronization link cannot be created between any two objects. The synchronization relationship must follow the composition definition of the object's class. An object window can be the owner of another window in the synchronization tree if, and only if, the object displayed in the owner window has an attribute that references the object displayed in the owned window.

The synchronization mechanism can be better illustrated through an example. Let **A** be a class with components **B** and **C**. Supose that the object **A1** of class **A** is presented in an object window. This window shows that **A1** has sub-objects that belong to classes **B** and **C**. If the user selects the sub-object of type **B**, the system creates an object window containing the object **B1**. In a similar way, the selection of the sub-object of type **C** causes the creation of an object window for the object **C1**. As a result of this process, it is created the synchronization tree showed in figure 8.

Let **A2** be the next object of **A**, obtained through the application of the *next* operation in the object window of **A1**. At the same time of the selection of this sequencing operation, due to the synchronization tree, the sub-objects of **A2** (respectively **B2** of type **B** and **C2** of type **C**) will be displayed in the two remaining windows, automatically (figure 9).

Therefore, a single sequencing operation can update the presentation of several objects, through the synchronization mechanism. It is important to note that the predicates defined for each object window continue to be verified when the object window is synchronized with other windows. The association of predicates with the synchronization mechanism provided by GOODIES is similar to a query processing facility where the user selects and restricts the required information. Only graphical query tools provide this facility, and none of the systems cited in section 1.1 have such a powerful mechanism for browsing.

## 6 Other Facilities

Besides the facilities for navigation and querying presented above, GOODIES provides many facilities that were developed in order to allow the user to customize the system according to his needs. These facilities are also important for adjusting the features of the system to accomplish some specific tasks.

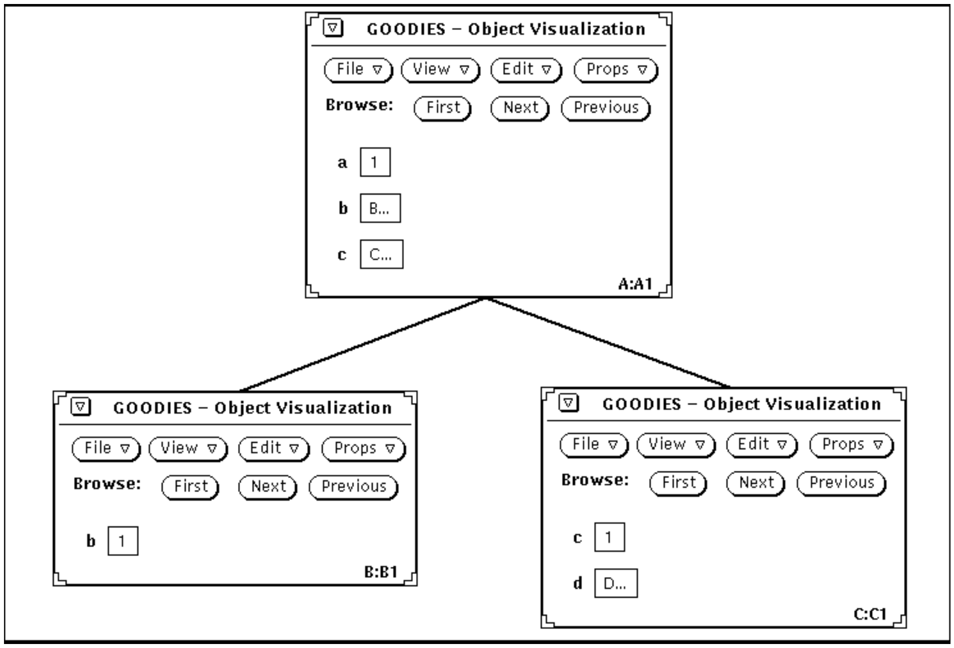


Figure 8: Synchronization Tree

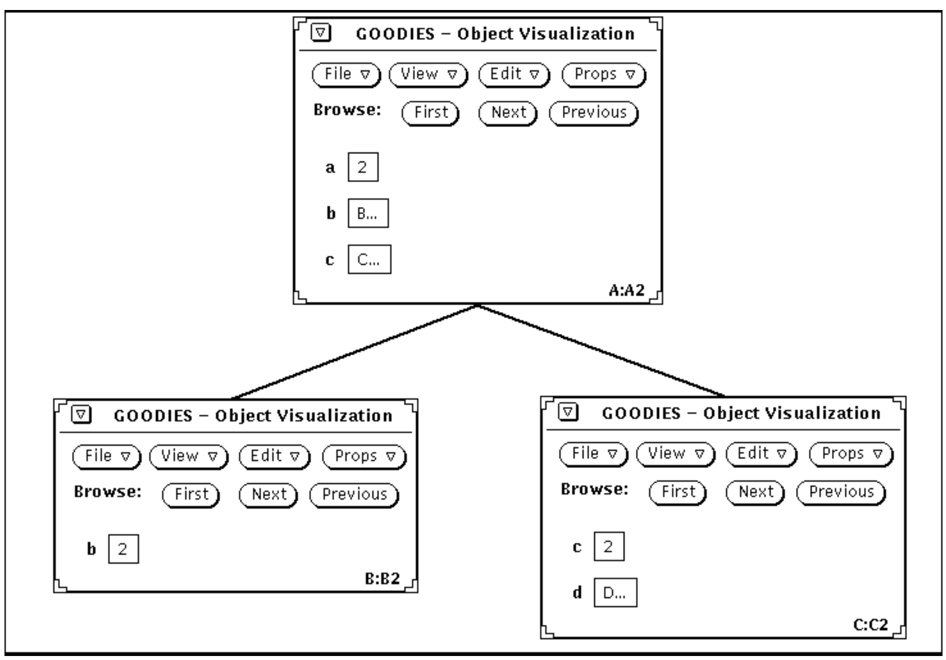


Figure 9: Synchronization Tree after *next* operation

### 6.1 Context Saving

The option *Save Workspace* associated to the *Props* menu of the DB window executes one of the customization functions available in the GOODIES system. This option tells the system to save the current workspace where the user is working. After saving his workspace, every time the user opens a GOODIES section, the system automatically presents the *context* that the user was visualizing in the moment he activated the *Save Workspace* option.

The term *context* is used here to denote the base windows (directory window, DB windows and class windows). The reason for the exclusion of object windows from the context is that objects are dynamically inserted in and removed from the DBs, whereas schemas are not expected to be modified often.

### 6.2 Visualization Level

According to the *inheritance concept*, the definition of a class inherits methods and attributes from its superclasses. Besides that, the inheritance hierarchy may have an arbitrary depth. If multiple inheritance is supported, a class inherits methods and attributes from all its superclasses. In this way, the definition of a class type may contain few attributes and methods defined for the class, with a large number of inherited attributes and methods. It may be the case that the user does not want to see the complete set of attributes and methods, but only part of them. GOODIES offers facilities to define the visualization level of the class hierarchy. The user can select the desired visualization level through the following options:

*Display Superclasses:* an auxiliary window containing the list of superclasses of a given class, and the user selects from this list the superclasses whose attributes and methods should be displayed. This selection updates the contents of the class window items *Type*, *Superclasses* and *Methods*, as well as the attributes visualized in the object windows that belong to the class.

*Display Subclasses:* in a similar way, the user can select the subclasses that he wants to visualize, starting from a given class window. The unselected subclasses are eliminated from the class subclasses list of the class window, and the instances of those subclasses are eliminated from the class objects list.

### 6.3 Attribute Selection

In an object-oriented database, the type definition of a class may contain an arbitrary number of attributes. Choosing the class hierarchy visualization level is often not enough to fill the users needs, since a class that has no superclasses can still have an excessive number of attributes explicitly defined for it.

GOODIES allows, through an auxiliary window associated to the object window, to choose the attributes to be displayed. The attribute selection window contains a list of all attributes defined for (and inherited by) the object, according to the current visualization level. Only the attributes selected in the attribute selection window are presented in the object window.

In a similar way the user can choose the items that are presented in a class window. As it was observed in section 3.1, the items that compose the class window are five: the textual definition of the class composition (*Type*) and four lists ((*Superclasses*, *Subclasses*, *Methods* and *Objects*).

## 7 Conclusion

We presented the design of the GOODIES system. The basic mechanisms for interaction with the user was described, and it was noted that many of the system functions were adapted from previously developed interface systems. In fact, one of the main advantages of the GOODIES system is to provide the best browsing functions from previous database interfaces assembled in one single tool.

The window construction style follows the OpenLook [Sun90] guidelines, and the concept of dividing a complex object representation in several windows was taken from [MSB90]. According to that work, the natural trend to depict complex information in a single representation is not always possible, besides being frequently inefficient. In general, it is better to display complex information (for instance, objects in a OODB) in more than one presentation, where each presentation is tuned to a particular aspect of the global information.

The basic GOODIES navigation mechanism was inspired on the database interface system described in [RC88], though the data model of this system is the Entity-Relationship, whereas GOODIES uses the object-oriented data model. The idea of synchronized browsing was strongly influenced by the KIVIEW system concepts, introduced in [MDT89].

The idea of using predicates to improve the navigation process is used in graphical query systems, and the GOODIES definitions of predicates is very similar to the concepts used in the PICASSO [KKS88] system, a graphical query system for the universal relation data model.

Finally the items that define a class window are similar to those used in the OOPE system, described in [Alt90b]. It should be noted, however, that none of the interface systems from which GOODIES *inherited* features are independent from their respectives DBMSs. This important characteristic, the independency from an specific DBMS, differentiates the GOODIES system from the interface that influenced its design.

At the present moment, all the functionality described is implemented in a prototype that uses a SUN SPARCstation as platform, under the UNIX operating system. The system applies the graphical resources of the XVIEW toolkit [Hel90], and contains about fourteen thousand lines of code written in C++.

## References

[ABD+89] Malcolm Atkinson, François Bancilhon, David DeWitt, Klaus Dittrich, David Maier, and Stanley Zdonik. The Object-Oriented Database System Manifesto. In *Proceedings of the First International Conference on Deductive and Object-Oriented Databases*, pages 40–57, Kyoto, Japan, December 1989.

[AGS90] R. Agrawal, N. Gehani, and J. Srinivasan. Odeview: The Graphical Interface to Ode. In *Proceedings of the 1990 ACM SIGMOD International Conference on Management of Data*, pages 34–43, Atlantic City, USA, May 1990.

[Alm91] Jay Almarode. Issues in the Design and Implementation of a Schema Designer for an

OODBMS. In *ECOOP'91 Proceedings*, pages 200–218, July 1991.

[Alt90a] Altaïr. The Looks Programmer Manual. Technical Report, Gip Altaïr, January 1990. Printing Revision 1.1, 9/01/1990.

[Alt90b] Altaïr. OOPE: The Object-Oriented Programming Environment. Technical Report, Gip Altaïr, January 1990. Printing Revision 1.1, 9/01/1990.

[BH86] Daniel Bryce and Richard Hull. Snap:A Graphics-based Schema Manager. In *IEEE Proceedings of the International Conference on Data Engineering*, Los Angeles, USA, February 1986.

[BM91] Elisa Bertino and Lorenzo Martino. Object-Oriented Database Management Systems: Concepts and Issues. *IEEE Computer*, 24(4):33–47, April 1991.

[BMP+92] P. Borras, J. C. Mamou, D. Plateau, B. Poyet, and D. Tallot. Building User Interfaces for Database Applications: The O2 Experience. *SIGMOD Record*, 21(1):32–38, March 1992.

[Cat91] R. G. G. Cattell. Next-generation Database Systems. *Communications of the ACM*, 34(10):30–33, October 1991.

[Com90] The Committee for Advanced DBMS Function. Third-Generation Database System Manifesto. *SIGMOD Record*, 19(3):31–44, September 1990.

[GGKZ85] Kenneth J. Goldman, Sally A. Goldman, Paris C. Kanellakis, and Stanley B. Zdonik. ISIS: Interface for a Semantic Information System. In *Proceedings of the 1985 ACM SIGMOD International Conference on Management of Data*, Austin, USA, May 1985.

[Hel90] Dan Heller. *XVIEW Programming Manual*, volume 7 of *The X Window System Series*. O'Reilly & Associates, April 1990. Second Printing.

[Jac91] Mike S. Jackson. Tutorial on object-oriented databases. *Information And Software Technology*, 33(1):4–12, January 1991.

[KKS88] Hyoung-Joo Kim, Henry F. Korth, and Avi Silberschatz. Picasso: A Graphical Query Language. *Software Practice and Experience*, 18(3):169–203, March 1988.

[Mam91] Jean-Claude Mamou. *Du Disque à le ecran: Genération D'Interfaces Homme–Machine Pour Objects Persistants*. PhD thesis, Université de Paris - Sud - Centre d'Orsay, May 1991.

[MDT89] Amihai Motro, Alessandro DÁtri, and Laura Tarantino. The Design of KIVIEW: An Object-Oriented Browser. In Larry Kerschberg, editor, *Proceedings of the Second International Conference on Expert Database Systems*, pages 107–131. The Benjamin/Cummings Publishing Company, Inc., 1989.

[MSB90] John Alan McDonald, Werner Stuetzle, and Andreas Buja. Painting Multiple Views of Complex Objects. In *OOPSLA'90 Proceedings*, pages 245–257, Ottawa, Canada, October 1990.

[MvD91] Aaron Marcus and Andries van Dam. User-Interface Developments for the Nineties. *IEEE Computer*, 24(9):49–57, September 1991.

[Oli93] Juliano Lopes de Oliveira. Uma Ferramenta Gráfica para Navegação e Consulta em Bancos de Dados Orientados a Objetos. Master's thesis, Universidade Estadual de Campinas - Departamento de Ciência da Computação, March 1993.

[PH91] Thiagarajan Palanivel and Martin Helander. Human-Factors Issues in Dialog Design. *Advances in Computers*, 33(1):115–171, 1991.

[RC88] T. R. Rogers and R. G. G. Cattell. Entity-Relationship Database User Interfaces. In Michael Stonebraker, editor, *Readings in Database Systems*. Morgan Kaufmann Publishers, Inc., 1988.

[Shn83] Ben Shneiderman. Direct Manipulation: A Step Beyond Programming Languages. *IEEE Computer*, 16(8):57–69, August 1983.

[Shn87] Ben Shneiderman. *Designing the User Interface: Strategies for Effective Human-Computer Interaction*. Addison-Wesley, 1987.

[Sun90] Sun Microsystems. *OPENLOOK - Graphical User Interface Applications Style Guidelines*. Addison-Wesley, June 1990. Third Printing.