A Graphical Interface for an Object-Oriented Query Language

Herman Lam  H. More Chen  Frederick S. Ty
Jiwen Qiu  Stanley Y.W. Su

Depts. of Electrical Engr. and Computer and Information Sciences
University of Florida, Gainesville, FL 32611

Abstract

For database applications supported by a conventional DBMS, the database is generally defined using a data definition language and accessed using some data manipulation or query language. Although adequate for DBMSs based on simple models like the relational model, the language approach is no longer adequate or desirable for a DBMS that is based on a more powerful and complex semantic or object-oriented model. In this paper, a graphical interface for object-oriented query language (GOQL) is presented. GOQL is a part of a prototype knowledge base management system which is based on an object-oriented semantic association model, OSAM*. The object-oriented nature and the increased semantics of the underlying model and query language pose new challenges in user interface design due to their added complexity. On the other hand, these features also provide more information to the system in order to make the user interface more intelligent.

1. Introduction

The need for data models that are rich in semantics has become apparent in recent years as database management moves beyond simple models like the relational model to support engineering, scientific, statistical, and military applications. The limited semantic constructs in conventional data models such as the relational model are no longer adequate. As a result, much work has been devoted to the development of semantic data models [e.g., CHE76, SM77, HAM78, SU83, ABI84, HUL86, LOO86]. Also, in recent years, the importance of the object-oriented (O-O) approach to data modeling has been established. Several of the proposed semantic models have taken the object-oriented approach [COPU, ABI84, MAI86, ZDO86, SU89]. The O-O paradigm offers several features that are important in modeling data: object identity, data abstraction, encapsulation, and inheritance of structural and behavioral properties.

For database applications supported by a conventional database management system (DBMS), the database is generally defined using a data definition language (DDL) and accessed using some data manipulation or query language. Although adequate for DBMSs based on simple models like the relational model, the language approach is no longer adequate or desirable for a DBMS that is based on a semantic data model or an O-O model due to its added constructs and complexity. As the underlying data/semantic models become more powerful and expressive, database management systems (DBMSs) with more power and intelligence can be developed to provide more functionalities to the user. However, the value of such a DBMS is limited if it is not usable by a large class of users with varying capabilities. In other words, it can be said that a DBMS is as good as its interfaces.

Research on user interfaces has been concentrating on reducing the burden on the user through visual aids. Some works have been done on graphical query interfaces for the relational model [MCN75, HER80, STO82]. However, most of the graphical interfaces were developed for semantic data models. In particular, much work has been done in support of the Entity-Relationship (E-R) model [CHE76], including the Entity-based User Interface (CAT80), a Graphical User Interface for Database Exploration (GUIDE) [WON82], a Graphical Query Language for ER Databases (ZHA85), the "Living in the Database" (LID) system [FOG84], a Graphical Query Facility for ER Databases (ELM85), a Graphic Query System (QBD) [CAT88], and an Icon Presentation Module for an ER Database (SICON) [GRO88].

Graphical query interfaces for other semantic data models include SNAP system [BRY86] for the IFO model [ABI84], SKI [KIN84] and ISIS [GOL85] for the Semantic Data Model [HAM78], the Graphical Query Language for the Binary Relationship Model [MAR87], and GQL [DUB90].

At the Database Systems R & D Center at the University of Florida, a project is in progress to implement a knowledge base management system (KBMS) [SU88, LA889] using an object-oriented semantic association model (OSAM*) [SU89] as its underlying model. Within that context, this paper describes the design and implementation of a Graphical user interface for an Object-oriented Query Language (GOQL). GOQL consists of a Graphical Browser which allows a user to browse through a complex knowledge base schema graphically, without having to wade through pages of textual DDL description of the schema. The Browser contains a set of useful features: zoom/unzoom, hide/show classes, hide/show associations, show toplevel design, show/hide next level of details, help/instruction messages, etc. These features are provided for the user to "prune" a complex schema into a desired level of abstraction before the querying process. The user can then interactively and graphically construct queries, without having to learn and remember the syntax and semantics of the query language.

The work reported here is built upon some excellent works on graphical interfaces referenced earlier in this section. Although all graphical interfaces share some common features, they are greatly influenced by and highly dependent on the underlying models and languages. The work described here is different from existing interfaces in that the object-oriented nature and the increased semantics of the underlying model and query language pose new challenges in user interface design. On the other hand, these features also provide more information to the user interface and thus more opportunity to use this information to make the interface more powerful and intelligent.

The remainder of this paper is organized as follows. Section 2 provides a brief description of the underlying object-oriented semantic association model (OSAM*) and the object-oriented query language (OQL). In Section 3, an overview of a prototype knowledge base management system and the associated set of user interfaces and tools are first given. Then the design and implementation of the key components of GOQL (the Browser and the querying module) are described in detail. In Section 4, a summary and some comments on future work are given.

* This project is supported the Florida High Technology and Industrial Council grant #UPN-88092237 and the National Science Foundation grant #DDM-8814989.
2. An Object-oriented Semantic Association Model and Query
Language

2.1 An Object-oriented Semantic Association Model (OSAM*)

The Object-oriented Semantic Association Model (OSAM*) [SU89] provides a conceptual basis for uniformly capturing the semantics and inter-relationships among objects in an application world. Objects in the application worlds are modeled as OSAM* objects and grouped together into classes based on some common semantic properties. The expressive power of OSAM* lies in its capability to use various semantic association types in a nested and recursive fashion to model application-world objects and semantic relationships among these objects. In the core OSAM* model, there are five system-defined semantic association types: Aggregation (A), Generalization (G), Inheritance (I), Composition (C), and Association (X). User-defined association types are also allowed. An example shown in Figure 1 is used to illustrate some of the key OSAM* concepts, particularly the A, I, G, and C association types.

Shown in Figure 1 is a part of an educational database modeled by OSAM* in the form of a semantic diagram (S-diagram). An S-diagram is an OSAM* graphical representation of classes and their associations by means of a network of labeled nodes and edges. Each rectangular node in the S-diagram represents an entity class (E-class), which models objects that are accessed independently. Objects in an E-class are uniquely identified by system-assigned object identifiers (OIDs). A circular node represents a domain class (D-class), which models objects that primarily serve as descriptive data of some other object. Objects in a D-class (e.g., integers, characters, etc.) are identified by values. A node that represents a class being defined (called defined class) is connected to its constituent classes via labeled edges. These edges are grouped together if they enter into the same association with the defined class. They are labeled by a semantic association type (e.g., A, I, G, C, or X).

Edges that are labeled by A and I represent the attributes of a defined class. The constituent classes to which these edges point are the domains from which the attributes draw their values. For example, the E-class Course has aggregation (A) associations with two constituent D-classes (CourseNum and Title) and a constituent E-class (Department). These associations define the three attributes for Course: Department, CourseNum, and Title, which draw values from the three constituent classes. An attribute has the same name as its domain unless specified otherwise. In the latter case, the edge is explicitly named in the S-diagram, such as the edge named "Major" connecting Student and Department. Also, an attribute or a composite of attributes may be designated as the user-specified identifier of an E-class. Its values uniquely identify the objects of the E-class. For example, the dash-marks on the edge CourseNum of the class Course designates that attribute as the user-specified identifier of Course. Note that the aggregation association is similar to the concept of "instance variables" in a conventional O-O system.

The E-class Student has two types of associations: generalization (G) and aggregation (A). The generalization association is similar to the superclass-subclass concept of object-oriented systems. In this case, the class Student is a generalization of the classes Grad and Undergrad. In other words, all graduate students are students and all undergraduate students are students. The set-exclusion (SX) between its constituent classes specifies that a student cannot be a graduate and an undergraduate student at the same time. Set-intersection, set-equality, and set-subset are the other constraints.

The aggregation associations of the class Student models the semantics that the attributes, GPA and Major, characterize the objects of Student. Since a undergraduate student is also a student in the application domain, an instance of Undergrad inherits the two attributes of the more generic class Student. Additionally, the Undergrad objects themselves have the attribute Minor that is specific to Undergrad. Similarly, an instance of Grad has the attributes GPA and Major inherited from Student. In general, a class can be a superclass of many classes and a subclass of many other classes, resulting in a generalization (G-) lattice. Thus, instances of a subclass in an OSAM* database inherit all the associations, operations, and rules from its superclass(es).

The E-class Transcript has an interaction (I) association and an aggregation (A) association. The interaction association models the concept that an object in this class represents a relationship between or among some objects in its constituent classes. In this case, each instance of Transcript represents the fact that a student has taken some course. This interaction is described by the attribute Grade. The many-to-many (or n:m) cardinality constraint between the constituent classes is depicted on the arc. The interaction association is similar to the "relationship" concept in [CHE79].

The above description provides the reader with an overview of OSAM* so that examples given in subsequent discussions will be meaningful to the reader. For a more detailed and formal treatment of the semantic associations of OSAM*, refer to [SU89].

2.2 An Object-oriented Query Language (OQL)

Similar to a query language such as the relational SQL, the Object-oriented Query Language (OQL) is a high-level query language that facilitates the associative access and manipulation of data in a database [ALAB89]. However, unlike SQL access to a database is not conceptualized as the "joining" of tuples or records from a collection of relations by specifying the relationship between keys and foreign keys and then "projecting" and "selecting" the desired data. Instead, an OSAM* database is visualized as a collection of inter-related objects which are grouped into classes. The semantics of the database is described by a schema and graphically represented by an S-diagram. Access to such a database using OQL is conceptualized as a two-step process:

1) establishment of a database "context" for the query which defines a subdatabase containing a subset of the original objects and object associations. This is a "filtering" process to relate and retain only the relevant parts of the database to be further processed.

2) specification of the operation(s) to be performed on the identified object(s) of the class(es) in the context by message passing. The general syntax of an OQL query block is as follows:

```
CONTEXT association pattern expression
WHERE conditions
SELECT object classes and/or attributes
Operation parameter list
```

The CONTEXT clause and its optional subclauses (WHERE and SELECT) are used to systematically restrict the retention of the qualified objects and their associations in the desired subdatabase. A typical association pattern expression has the form:

```
Class[| intra-class-condition] op Class[| intra-class-condition] op ...```

Each "Class" specifies one of the classes from the original database to be retained in the subdatabase. The intra-class condition is used to retain those objects in each specified class that satisfy certain conditions. "Op" is one of the association pattern operators (to be illustrated below) used to further restrict the subdatabase and to retain only those objects from different classes that are related in a prescribed manner. The optional WHERE subclause is used to specify inter-class conditions to further eliminate unqualified objects from the subdatabase. It allows objects in different classes to be arbitrarily related through their attribute values. The optional SELECT clause is used to "project" the subdatabase over some classes and attributes to retain only the relevant data for further processing (e.g., display).
The Operation clause specifies the message to be sent to the identified objects. It contains the operation name and a parameter list. Included in the parameter list is the class(es) in the subdatabase to which the message is to be sent. The operation can be a system-defined operation (e.g., Display, Update, Delete, etc.) or a user-defined operation (e.g., Rotate, OrderPart, AssignAdvisor, etc.).

An example is now given to illustrate some of the concepts in OQL, including the association (*) and the non-association (!) operators.

**Query:** For departments which have graduate courses (c-no >= 5000) that are not being offered currently, display the department name and the course title.

After studying the S-diagram shown in Figure 1, we see that this query involves three classes (Departments, Course, and Section). Figure 2(a) shows a part of the university database (the part concerning these classes) at the instance level in the form of an object diagram. Each dot represents an object instance. Object instances of a class are grouped together. The links among object instances represent the associations among them. For example, it is shown that Department object d1 is associated with Course object c1, which is associated with Section object s1. This represents the fact that department d1 has a course c1 which is currently offered as section s1. Similarly, we can see that department d5 has 4 courses, one of which is offered as one section (c7, s4), one of which is offered as multiple sections (c6, s2, s3), and two of which are not offered currently (c5, c8). Thus, in general, a department can have 0 (e.g., d3) or more courses and each course may be offered as 0 or more sections.

To satisfy the query in Example 1, we develop the OQL CONTEXT expression as follows:

1. Only those Department objects and Course objects which are associated with each other (i.e., Department which have courses) are retained. We can represent this semantics in OQL with the association operator (*):

   ```
   CONTEXT Department * Course
   Note that the Department object d3 will be eliminated.
   ```

2. However, only graduate courses are of concern. Thus the CONTEXT expression is refined:

   ```
   Department * Course[c-no >= 5000]
   ```

3. Finally, only care about those courses that are not being offered. This can be expressed in OQL using the non-association operator (!):

   ```
   CONTEXT Department * Course[c-no >= 5000] ! Section
   ```

Graphically, the CONTEXT produces the subdatabase shown in Figure 2(b). Note that only the departments which have (graduate) courses that are not currently offered are retained. For example, if c2 is not a graduate course, then c2 (and d2) will not be retained.

To complete the OQL query, the SELECT clause and the Display operation are specified:

```
SELECT Department, Course[name], Course[title]
Display Department
```

The object-oriented query language (OQL) is a high-level query language that facilitates the associative access and manipulation of data in an O-O database through the use of various association operators. These association operators allow complex association patterns to be specified in a simple way. Two such operators were illustrated by this example. This example provided a brief overview of the object-oriented query language so that it can be used to illustrate the graphical interface GOQL to be described in the next section. For more details concerning OQL, refer to [ALAS9].

**3. A Graphical User Interface for an Object-Oriented Query Language**

GOQL is a part of a prototype knowledge base management system developed at the University of Florida. A block diagram of the prototype knowledge base system is shown in Figure 3. The prototype system consists of two main components: a knowledge base management system (KBMS) and a set of user interfaces and tools. The KBMS is based on OSM* and itself consists of two main components: an object manager which provides low-level functions to store, access, and manage the objects stored in the knowledge base; and a set of software which provide the actual database and knowledge management functionalities. A user of this system can interact with the KBMS through a variety of languages, user interfaces, and tools including a knowledge base definition language (KDL) [PUR88], an object-oriented query language [ALAS9], a schema design tool [DSO88, PAN89], and GOQL. In the remainder of this section, the design and implementation of GOQL will be described.

The prototype system has been implemented on a SUn3 workstation. The object manager is implemented "on top of" Vbase, an object-oriented DBMS developed by Ontologic, Inc. The knowledge base software is implemented in C. The graphics user interfaces and tools are implemented in C with the support of the Sunview graphics package.

**3.1 GOQL Requirements**

Based on the experience of the existing works on graphics interfaces referenced in the introduction section and our own work [DSO88, TY88, PAN89], it is clear the following features are necessary in the development of a graphical query interface for semantically rich data model:

1. The querying tool should provide a mechanism for the user to interactively browse and understand the database to be queried. It should not assume that the user has total understanding of a complex schema and has committed it to memory.

2. The browser should present a graphical representation of the schema. The user should not have to wade through pages of textual DDL description of a complex schema (on-screen or off-line) trying to understand and visualize the relationship among the classes.

3. The user should be allowed to browse and query at a level of abstraction required by the user for the current querying session. Classes, associations, and other details not required for the current query should be allowed to be "hidden".

4. The user should not have to memorize the syntax of the query language. The tool should be interactive, with constant guidance and information feedback at each stage of the querying process.

5. The querying tool should prevent errors as much as possible. For example, typing should be minimized and the tool should refuse to accept erroneous inputs, etc.

6. The tool should have different modes for users of different capabilities. Novice users should be provided with more guidance and explanation; whereas, advanced users should be given more flexibility. Also, the tool should help a user improve his/her querying capability.
through the use of the tool.

GOQL posses all the above features.

3.2 Graphical Browser

The Graphical Browser is one of two main components of GOQL and is designed to satisfy the first three requirements stated in the previous section. The Browser contains three key windows: schema window, command window, and message window. The schema window displays the schema of interest, representing it graphically in the form of an OSAM* semantic diagram. The command window, to be described below, provides a set of browser commands in form of "buttons". The message window is used to provide help, error, warning, and status messages to the user during the browsing process. The main features of the Browser are represented by the command buttons in the command window. Functionally, they can be categorized as schema traversal commands or schema pruning commands. These functions are described below.

A large complex schema is logically divided into many pages. The schema can be displayed in the schema window one page at a time. Each schema page is larger than a screen that can be displayed on the schema window. Thus, scroll bars can be used to move the schema window to display the different parts of a schema page. The <Unzoom> command allows the entire page to be shown at once. To move between pages, the following command buttons can be used. The pages are numbered sequentially. The <Former Page> and <Next Page> commands bring in for display the previous and next page, respectively. The <Page> command allows the user to specify a particular page number to display that page. The <Focus> command allows the user to specify a class name. The system will search for that class and the corresponding page will be brought in and displayed with the focused class in the center. Furthermore, the user can click on an "intelligent" connector and choose the "Go to class" selection. The system will search for the class pointed to by the connector and bring in the corresponding page for display.

The remaining commands are used to prune the schema in preparation for the querying process.

- <Hide/Show Attrs> - By default the names of attributes (if they are different from their domain names) are displayed. If desired, the user can hide the display of them in the schema window.

- <Hide Domains> - This command hides the display of all the domain classes. This is useful if the user want to display only the entity classes without the cluttering details.

- <Hide Classes> - This command allows the user to hide specific classes of his/her choice.

- <Hide A's>, <Hide G's>, <Hide S's>, <Hide C's>, <Hide X's> - These commands can be used to hide association links of a specific type. For example, if a user is interested in viewing only the generalization hierarchy or lattice, he/she can hide the other association types.

- <Show TopLevel> - This command displays only the part of the schema that is designated as the top-level design. The top-level design consists of the key classes, associations, operations, and rules (as designated by the schema designer) that comprise the 'essence' of the schema, without the cluttering details. For example, a new user of a complex schema should begin the browsing by using this command.

- <Show/Hide NextLevel> - This command can be used to increase or decrease the amount of details to be viewed by showing or hiding the next level of details "below" a particular class.

- <Show All> - This command will restore the schema to its original state.

In addition to these schema-level browsing and pruning facilities, browsing into the details of a class can be done. When a user click at an object class in the S-diagram, a pull-down menu will be displayed. The details of a class which can be browsed include the textual DDL definition of the class, the methods and rules defined for the class, and the result of a query which involves this class.

After having browsed the S-diagram to gain a clear understanding of the part of the database to be queried and pruned the schema to a level of abstraction suitable for this query, the user can begin the querying process.

3.3 Querying Module in GOQL

Shown in Figure 4(a) is a screen set-up for the querying module in GOQL. It consists of four windows: schema window, command window, message window, and OQL window. As in the Browser, the schema window displays the schema of interest, now pruned to the desired level of details during the browsing session. The command window, to be described below, presents the available commands at each level of the querying process. The message window is used to provide guidance, error, and feedback messages to the user. The OQL window is used for two purposes: 1) to directly enter a textual OQL query if desired, and 2) to display the textual OQL equivalent of the graphical GOQL query at each stage of the querying process.

3.3.1 Querying in GOQL

In GOQL, there are two querying modes, "OQL" and "Graphical OQL". The "OQL" mode is provided for knowledgeable users who do not need the help nor want to be burdened with the various levels of menus offered by the graphical OQL mode. In the OQL mode, the user can directly type in the OQL command in the OQL window.

In the graphical OQL mode, GOQL guides the user through the following steps in the construction of a query:

- **CONTEXT formation.** In this step, the user is asked to choose (with the mouse) the classes of interest and specify the relationship among them (using the operators: *, 1, *AND, etc.). Class search conditions can be specified at this step. The user is guided through the formation of a CONTEXT expression of any complexity that is allow in OQL (e.g. linear expressions, multi-level AND/OR branchings, loops, etc.).

- **WHERE clause.** After the CONTEXT formation, the user is allowed to specified an optional inter-class condition. This is the WHERE clause in OQL.

- **SELECT clause.** At this step, the user is asked to select the classes and attributes to be retained in the resultant subdatabase.

- **Operation.** Finally, the user is asked to choose the operation for the query. The operations can be system-defined operation (e.g. Display) or a user-defined operations that is defined for a class which has been retained in the subdatabase.

We shall use the example query from the previous section to go through a GOQL session to give the reader a feel of the system.

The user begins by invoking the Browser to browse through the schema and to focus on the part of the schema to be queried. Suppose that the university schema is pruned using the Browser to that shown in Figure 4(a). The first step in the querying process is to form the CONTEXT expression. The user is first asked to specify the E-classes of interest for the query. For the query, the user should select (using the mouse to click on the classes) Department, Course, and Section. At this point the schema window and the OQL window
appear as shown in Figure 4(b). Note that the selected classes are highlighted and the actual OQL expression is automatically constructed and shown in the OQL window. Also, note that the default association (*) operator is used between classes. As shown, the expression Department * Course * Section means to retain departments which have courses that are being offered as sections. To obtain the necessary CONTEXT expression for Query 1, the * operator between Course and Section needs to be changed to the non-association (!) operator. Also, we need to restrict the Course class to retain only graduate course (i.e., c_no >= 5000). These two tasks can be done using the <Special> command. When selected, the menu for the <Special> operations pops up. The user is guided to specify the intra-class restriction condition and to change the operator.

The graphical and the corresponding textual results of the CONTEXT formation step is shown graphically in Figure 4(c). At this point, the user sees graphically and textually that the context for the query specifies that the subdatabase contains Department objects and Course objects (those with c_no >= 5000) that are related (i.e., *) to each other, but not related (i.e., !) to any Section object.

Since there is no inter-class condition for this query, the user is directed to skip the WHERE clause formation. Next, the user is directed to select (using the mouse) the classes and attributes to be retained for the resultant subdatabase. Finally the system-defined Display operation is chosen. The resultant graphical and textual OQL query is shown in Figure 4(d). Note that the classes and attributes that are not a part of the query are no longer displayed.

4. Summary, Conclusion, and Future Work

In this paper, a graphical user interface for an object-oriented query language, GOQL, is presented. GOQL is a part of a prototype knowledge base management system which is based on an object-oriented semantic association model, OSAM*. It consists of a graphical Browser and a graphical querying module. The Browser allows a user to browse through a complex knowledge base schema graphically and then choose it into a desired level of abstraction and details before the querying process. In the querying module, there are two modes, OQL and graphical OQL. The OQL mode is provided for knowledgeable users to directly type in the OQL command. In the graphical OQL mode, the user is guided through the formation of the query. The OQL query is incrementally and graphically constructed. The process is interactive, with constant guidance and information feedback at each stage. A novice user needs not learn (and is in fact unaware of) the OQL syntax. Also, the textual OQL query is incrementally formed and shown in the OQL window, providing a novice user with a training tool to learn the OQL language.

Currently, our work on GOQL is concentrated in two areas. We are working on the integration of GOQL with the other user interfaces and tools into an integrated querying and development system. Also, we are working to make GOQL more "intelligent". Our approach is based on the fact that the underlying model (OSAM*) is an object-oriented knowledge representation model which is rich in semantics. Based on the additional semantics and knowledge about the data, the system can provide the user with "active" expert-system-like help in the construction of queries and the design of knowledge bases.

REFERENCES


Figure 1  S-diagram of University schema

Figure 2. Illustration for Example Query 1.

Figure 3. Overview of the Prototype System
Figure 4. GOQL illustration for Query