Implementation of the GOQL language.

Euclid Keramopoulos[1]
Philippos Pouyioutas[2]
Tasos Ptohos[3]

[1]Department of Informatics, T.E.I. of Thessaloniki, Greece
[2]Department of Computer Science, Intercollege, Nicosia, Cyprus
[3]Cavendish School of Computer Science

Implementation of the GOQL Language

Euclid Keramopoulos
Department of Informatics
T.E.I. of Thessaloniki
P.O. Box 14561
Thessaloniki 54101, GREECE
euclid@it.teithe.gr

Philippos Pouyioutas
Department of Computer Science
Intercollege
46 Makedonitissas Avenue
Nicosia 1700, CYPRUS
pouyioutas.p@intercollege.ac.cy

Tasos Ptohos
School of Computer Science
University of Westminster
115 New Cavendish Street
London W1W 6UW, UK
tasos@wmin.ac.uk

Abstract

The Graphical Object Query Language (GOQL) is a graphical query language that complies with the ODMG 3.0 standard and runs on top of the o2 DBMS. GOQL provides users with the User's View (UV) and the Folders Window (FW), which serve as the foundation upon which end-users pose ad-hoc queries. The UV is a graphical representation of any underlying ODMG scheme. Among its advantages is that it hides from end-users most of the perplexing details of the object-oriented database model, such as methods, hierarchies and relationships. To achieve this, the UV does not distinguish between methods, attributes and relationships, it encapsulates is-a hierarchies and it utilises a number of desktop metaphors whose semantics can be easily understood by end-users. The FW is a condensed version of the UV and provides the starting point for constructing queries. In this paper, we demonstrate the UV and the FW and discuss GOQL's system architecture, its various components and the way these components interact to generate the UV and the FW and to provide an ad hoc query construction mechanism. We also present the screen interface of the language.

1. Introduction

The evolution of database languages is strongly related partly to the evolution of user interfaces and partly to the evolution of database models and database systems in general. Before the early eighties not much attention was paid to the attractiveness/popularity or user-friendliness of user interfaces, mainly due to software/hardware limitation and the type of users expected to interact with computer systems. As processing power increased and the use of graphics was introduced to user interfaces, the type of the expected user of these systems changed from that of a highly skilled one to that of a computer literate (or naive) user. This has happened because of technological advances that resulted in an ever-increasing computing power that allowed computers to 'acquire' skills that users used to have; an immediate implication of this was a change in the users training needs. Nowadays, users instead of having to deal with the technical aspects of computer systems, need only to learn how to complete simple work tasks, whereas the problems they have to solve are usually expressed in non-computing terms. Furthermore, user interfaces are being designed according to users' skills, because designers believe that this is essential for improving users' productivity [1]. A detailed survey of the evolution of database query languages leading to the development of Graphical Query Languages (GQLs) can be found in [2].

Our research on graphical query languages has contributed to the subject area as follows. First of all, we designed a new graphical scheme representation, namely *User's View (UV)*, which has as basic characteristics the use of (a) human factors, like desktop metaphors and colour, and (b) the elimination of technical details without loosing anything of the database model power. The contribution of our research is that the UV is addressed to all types of users including naive ones and also, that it is designed to be independent of the underlying database model. Besides that, we designed a new graphical query language, namely *GOQL* (Graphical Object Query Language), which has the same expressive power as the ODMG 3.0 standard OQL [3] and it is the only GQL for the ODMG 3.0 that supports also binding functions and method parameters.

In this paper, in Section 2, we demonstrate, using an example, the UV and the FW and the way they support the construction of graphical queries. In Section 3, we discuss GOQL's system architecture, its various components and the way these components interact to generate the UV and the FW and to provide an ad hoc query construction mechanism. We also address issues related to the implementation of the GOQL system architecture and the portability of GOQL across different DBMS platforms. In Section 4 we present some of the screen interfaces of GOQL to demonstrate the implemented functionality of the system. We conclude by discussing our current and future work. The interested reader can find further information regarding the formal model, the design, the use, the implementation and the evaluation of GOQL in [2,4,5,6,7,8,9,10,11,12], whereas a detailed discussion of the advantages GOQL offers compared to the other graphical query languages can be found in [2,4,7].

2. The User's View (UV) and the Folders Window (FW) of GOQL

The GOQL was designed to address the needs of end-users and to provide an alternative graphical query language to OQL and ODBMSs that support OQL (e.g. o2). Thus, GOQL was designed to comply fully with the features of the object model of the ODMG 3.0 [3] and its query language, OQL. GOQL allows users to express graphically a variety of ad hoc queries ranging from simplistic ones to rather complicated ones. The features provided/supported by GOQL include: the support of a 2D colour interface, the use/support of methods, predicates, Boolean & set operators, arithmetic expressions, existential /universal quantifiers, aggregate functions, group by and sort operators, functions, and sub queries.

To achieve these, GOQL users are presented with the User's View (UV), which is GOQL's graphical representation of an underlying ODMG database scheme and which serves as the foundation upon which GOQL queries are constructed. The UV allows all the features of the underlying ODMG object model to be represented, but it hides from users most of perplexing details, such as methods, hierarchies and relationships. In particular, it

- does not distinguish between methods, relationships and attributes;
- does not support the explicit representation of is-a hierarchy lattices; instead it treats any inherited by a subclass properties as properties of the subclass itself and it represents them as such;
- utilises a number of desktop metaphors that allow the representation of the other features of the object model.

The UV is generated using the metadata of a given ODMG database scheme and it is comprised by a number of UV_class_tables. Each UV_class_table is a representation of a class of the given ODMG database scheme. It contains a representation for each of the attributes, relationships and methods, including any inherited ones, of the class that a particular UV_class table represents. More specifically, the following one-to-one mapping can be defined between constructs/features of the ODMG data model and the graphical representation of these constructs in the UV.

- Each class is mapped onto one UV_class_table.
- Each attribute of a class is mapped onto a property (row) of the corresponding UV_class_table.
- Each method of a class is also mapped onto a property (row) of the corresponding UV_class_table.
- Each relationship is mapped onto a property (row) that is linked to a folder or a briefcase. A folder is used to denote a class object, whereas a briefcase is used to denote a class that has subclasses. Thus, as far as users are concerned, there is no distinction between an attribute, a method and a relationship.
- Each inherited attribute/relationship/method of a class is mapped onto a property (row) of the corresponding UV_class_table. Thus, inheritance is hidden from users who see the properties of any superclasses as properties of the UV_class_table itself.
- Although types are hidden from users; i.e. properties do not display their type, the following two exceptions have been made as it was thought that the used denotations help users understand better the database scheme.
    - A structure type is represented by an envelope which a user can "open" to reveal the properties that constitute this structure type. The envelope is placed to next to the right hand side edge a property row.
    - a collection type, i.e. a set, a bag, a list or an array, is represented by a paper clip that is placed on the right of the top edge inside a property row.

In Figure 1, we give the UV of the o2 ODMG database scheme given in Appendix I. The UV shows all UV_class_tables, their properties and their relationships and we believe that it offers a much more simplified representation of the database scheme compared to the ODMG one.

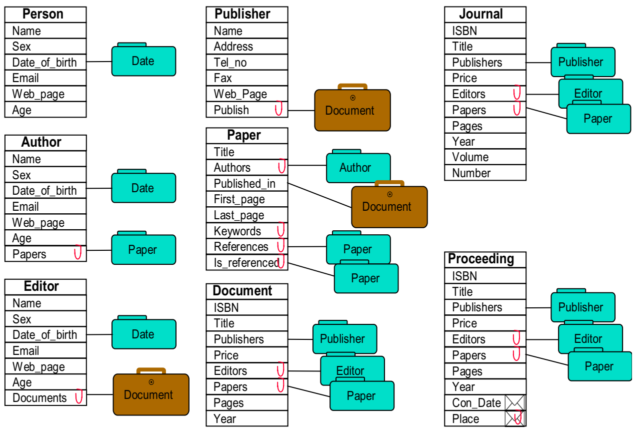


Figure 1: The User's View for the o2 ODMG Scheme

GOQL also provides a condensed UV, namely the Folders Window (FW), that consists only of the class tables represented as folders, i.e. no properties and/or relationships are being shown, see Figure 2(a). Users can "select" and "open" any of the folders contained in the Folders Window to reveal its contents. By "opening" a folder users effectively descend a level into the structure of the ODMG class that a folder represents; to graphically represent this, users are presented with a window that contains the UV_class_table of any opened folders - see Figure 2(b). Users can find more details about the structure of a particular class by relationship browsing, i.e. selecting and opening folders / briefcases that are linked to rows of opened UV_class_table that represent relationships, see Figure 2(c). Thus, using the relationship browsing users can navigate and develop the part of the schema required for a query.

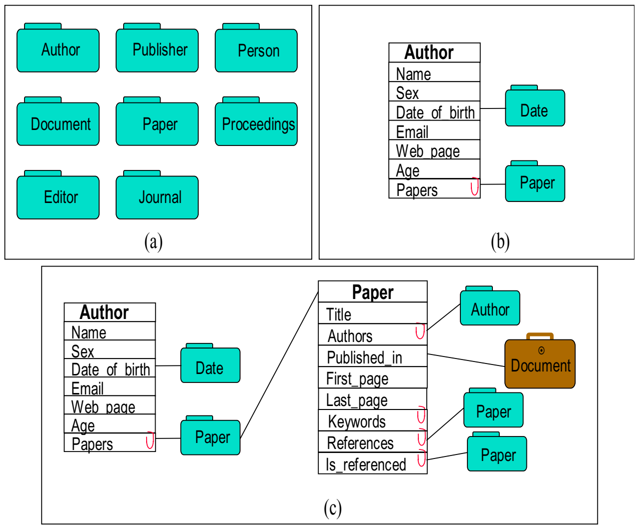


Figure 2: The Folders Window, Scheme Developing, and Relationship Browsing

3. GOQL's Architecture and Implementation

GOQL runs on the o2 Object-Oriented Database Management System [13] and was implemented using the o2 system and Tcl/Tk [14, 15]. In particular, Tcl/Tk was used for the implementation of the GOQL interface and the development of the GOQL translator. Tcl is an open source programming language that is based on the C programming language, whereas Tk is an open source language that provides developers of graphical user interfaces with a library of functions/tools that accelerate the development of graphical user interfaces. The open source nature of Tcl/Tk and the portability of Tcl/Tk code across platforms were the main reason that influenced our decision towards their use. The choice of the underlying OODBMS was determined by

(a) the support of the ODMG OQL and
(b) the availability of such DBMS.

The main reason for the ODMG OQL compliance requirement was the portability of the GOQL system. Thus, the o2 DBMS [13] was used as the underlying DBMS, whereas o2C [13] was used as means of passing the produced OQL query to the underlying OODBMS for processing and for handling the results returned. Although, GOQL was implemented based on the o2 DBMS, its design is such that it can be ported to another DBMS platform that complies with the ODMG 3.0 with minimal effort by making minimal changes to the way the data structure is generated from the metadata of the underlying OODBMS.

Figure 3, gives a pictorial description of the GOQL System Architecture. GOQL consists of the Metadata Translator, the Scheme Viewer, the Query Editor, the Error Handling Mechanism, the Help Mechanism and the Translator. Before any GOQL query is constructed and run, the relevant database is loaded in GOQL. The metadata of the underlying database schema (Appendix I) is provided from the OODBMs to the Metadata Translator, which constructs the database Data Structure (Appendix II). The Data Structure consists of a number of files that contain information about the scheme of the underlying DBMS. These files have the same structure regardless of the underlying OODBMs. Moreover the class files incorporate all the is-a hierarchies and don't distinguish between methods and attributes.

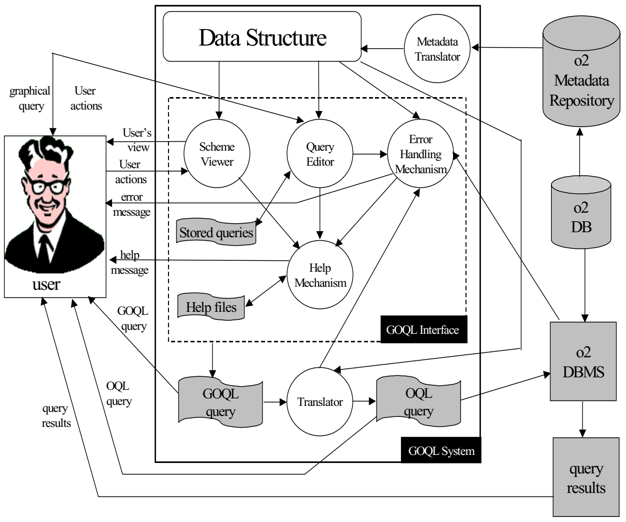


Figure 3: The Architecture of GOQL

Once the Data Structure is constructed, the Scheme Viewer is used to generate the User's View Window (Figure 1) and the Folders Window (Figure 2). The graphical representation of classes are based on the Data Structure class files and therefore incorporate the is-a hierarchies and do not distinguish between methods and attributes, thus hiding from users the perplexing details of the object-oriented database model and providing a simplified view of the underlying database schema. The UV and the FW provide the starting point for developing queries. In more details, a user can start constructing, loading, editing, deleting, running and storing graphical queries using the Query Editor (QE). The user can load folders from the FW in the QE and start opening them and developing the query. During the query development, the user can consult the UV in order to understand better the underlying database scheme. The new query can then be saved through the QE in the GOQL Data Structure. Saved queries can then be opened by the QE, edited and saved.

During the query development, the Error Handling Mechanism (EHM) checks for errors and informs the user accordingly. More specifically, the EHM checks for syntactical errors (incomplete or invalid graphical queries), compatibility errors and opening file errors (e.g. attempting to open a query before the database metadata is loaded, attempting to open a query for another database scheme, etc.). The Help Mechanism (HM) provides further general and/or specific information about GOQL and features thereof and suggests solutions for errors that may appear. The HM is available through the UV, the FW and the QE.

4. GOQL's Screen Interface

The first step, before any query is constructed in GOQL, is to load the underlying database metadata. This involves the use of a simple dialog box. Following the loading of the relevant metadata, users have the option to access the graphical scheme representation of the underlying database either in the UV or in the FW. These two forms are used throughout the query construction, either in consulting way for the database structure or for copying an object of a specific class on the Query Editor (QE) canvas. The QE is created either by opening a stored query or by starting a new query, by double clicking on an object appearing in the FW. The query construction takes place in the QEW using the available tools.

Starting the GOQL users are presented with the main GOQL window (Figure 4). The first step is to load the underlying database metadata. This involves a dialog process during which users choose from a list of available databases the one they require to use. Following their choice, users are presented with the FW and the UV.

Figure 4: The GOQL Main Window

To load the metadata users are presented with a dialog box (Figure 5), activated by selecting the *Open* option from the *File* pop-menu of the GOQL main window (Figure 4). The dialog box consists of three components, namely: the label, which displays at the top of the dialog box the path to the current directory; the 'UP' button, which users can use to move a level up in the directory structure; and the basic environment, which is comprised by two windows. The left window displays the names of directories defined within the current directory; by double clicking the left mouse on a directory name users can descend a level in the directory structure and move to the chosen directory. The right window displays all the o2 database files with the suffix '.load'. A user loads a database by double-clicking on the database name with the left button. Finally, a user can close a database and any open query window by using the *close* menu option of the *File* pop-up menu of the GOQL main window.

Following the selection of the required database users are presented with the FW. Users can also choose to display the UV. The FW contains all the objects of the chosen database. Each such object is represented by a closed folder icon (Figure 6). The FW is created either by opening a database or by selecting the *Folders* option from the *View* pop-up menu of the GOQL main window. Users can 'open' a folder by double clicking on it. Each 'opened' folder from the FW can become the root object for a subquery. Finally, whenever a user clicks the right mouse button on an object icon the name of that object class is displayed.

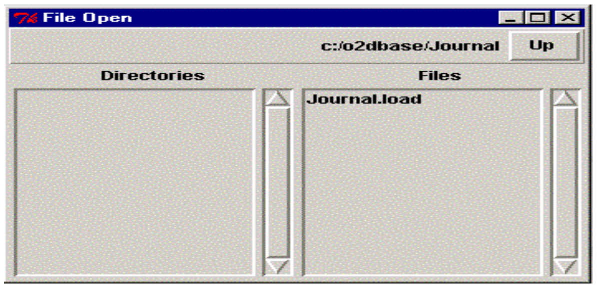


Figure 5: Load Database Dialog Box.

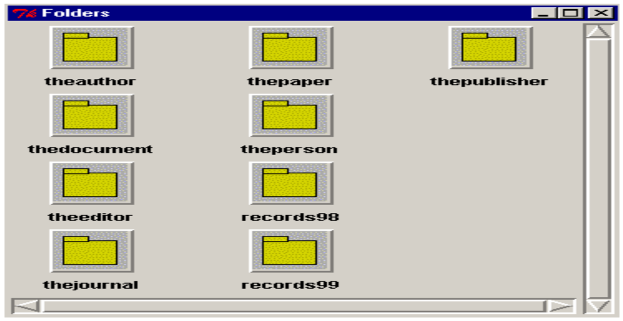


Figure 6: The FW

The UV Window (UVW) is created by selecting the *User's View* menu option from the *View* pop-up menu of the GOQL main window. Users can consult the UVW at any time during the query construction process. Figure 7 contains part of the UV of the running example.

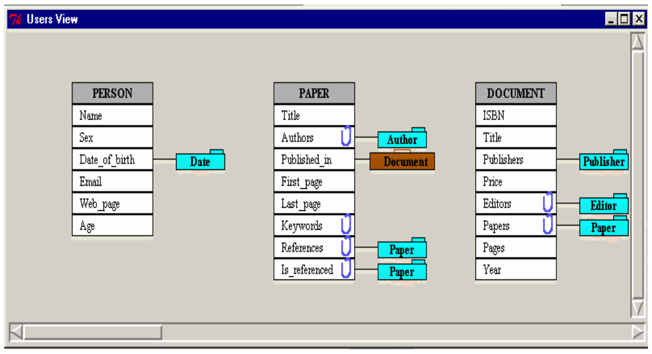


Figure 7: The UV Window (UVW)

The *Query Editor Window (QEW)* is the window where users can construct query expressions. Users are presented with a QEW if they select a root object from the FW or if they choose to open an already stored query expression. When the QEW opens, users are presented with a horizontal toolbar, a vertical toolbar, a menu bar, a canvas and a message line (Figure 8).

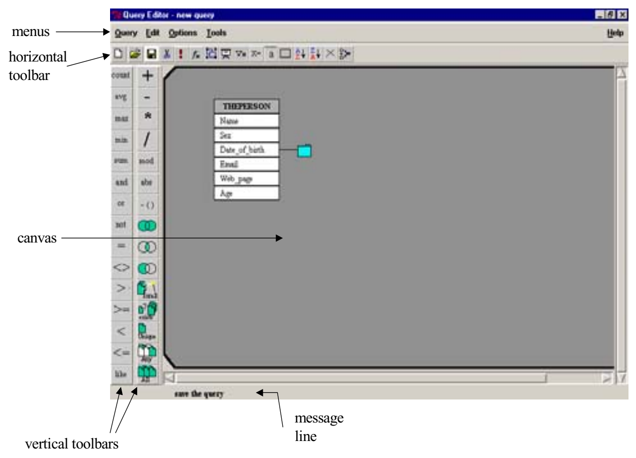


Figure 8: The Query Editor Window (QEW)

The canvas is the area where the query construction takes place and it is where the root object that was selected from the FW is displayed on. The message line is used by GOQL to display explanatory messages.

All the constructs/tools/functions that GOQL offers for the construction and manipulation of queries are organised and made available through the menu bar and the two toolbars. The menu bar contains five pull-down menus, namely the Query Menu, the Edit Menu, the Options Menu, the Tools Menu and the Help Menu. Selecting an option from any of these menus activates the action associated with the chosen option.

Both toolbars contain a set of buttons. Each has a unique icon and an action/tool associated with it. Each of the buttons corresponds to a menu option and they offer to users a faster way of invoking a particular action/tool than that of the pull-down menu. The icon of each button is a metaphor for the action/tool associated with the button. The metaphor used with each button has either been purposely designed or been selected because it has been commonly used in other well-established graphical user interfaces to represent the particular action/tool associated with this button. Selecting or 'pressing' a particular button involves placing the mouse pointer over it and clicking the left mouse button. Selecting a button activates the associated action/tool and makes the button appear on the user interface as being 'pressed'. Finally, whenever the focus of the mouse pointer is moved over a button the background colour of this button changes to white to highlight the event and an explanatory message about this button is displayed on the message line.

The horizontal toolbar, Figure 9, contains sixteen buttons, four or which (the cut button, the highlight button, the pick button and the missing button) have been designed for repeated use. Thus, when they are selected they remain selected until a different button is selected.

The vertical toolbar, consists of two columns each of which contains fifteen buttons (Figure 10). These buttons of the vertical toolbar are organised according to their functionality into nine groups. In particular, the aggregate functions group comprising the 'count', the 'avg', the 'max', the 'min' and the 'sum' button, the Boolean operators group comprising the 'and', the 'or', and the 'not' button, the comparison operators group comprising the '=', the '<>', the '>', the '>=', the '<', the '<=', and the 'like' button, the arithmetic operators group comprising the '+', the '-', the '*', the '/', and the 'mod', the absolute operator group comprising the 'abs' button, the negative operator group comprising the '-' button, the set operators group comprising the 'union', the 'intersect' and the 'except' button, the quantifying operators group comprising the 'for all', the 'exists' and the 'unique' button and the inclusive quantifying operators group comprising the 'any' and the 'all' button.

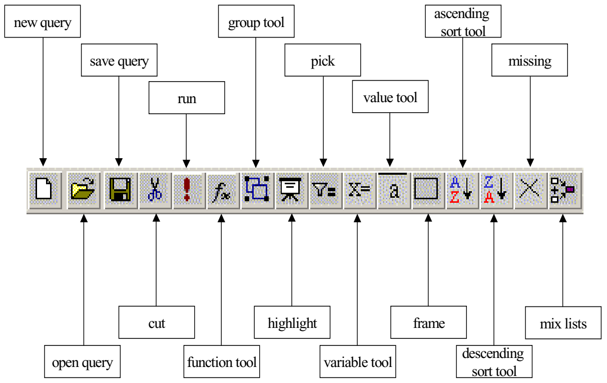


Figure 9: The Horizontal Toolbar

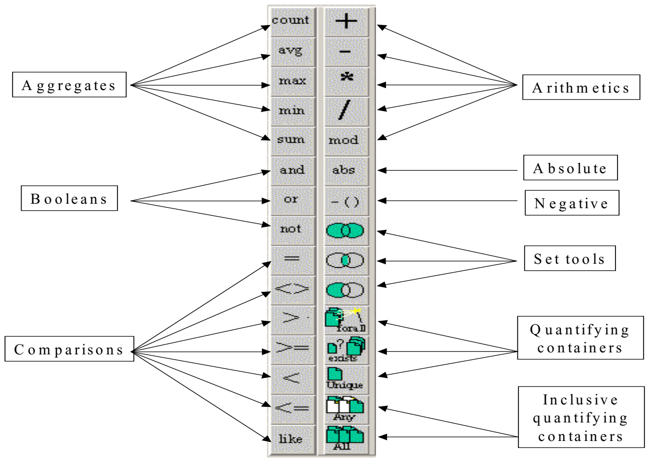
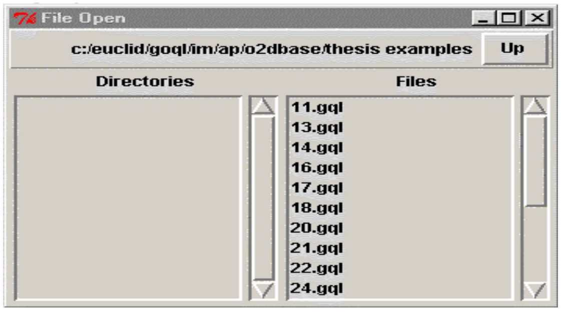


Figure 10: The Vertical Toolbar

The New Query button resets the query construction mechanism in order to prepare it for a new query (the path from the menu bar to the corresponding menu option is Query → New). If the button is pressed while a user is in the middle of a query construction, GOQL invokes the *save query* procedure before the 'new query' is activated. The starting point for a new query is always the FW where the user has to select the root object.

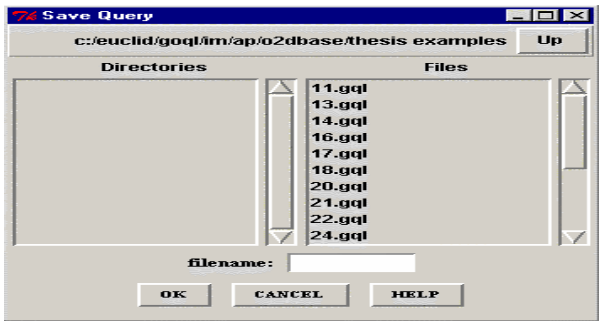
The Open Query button allows users to load a stored query expression in the QEW through the *file open query* dialog box (Figure 11) (the path from the menu bar to the corresponding menu option is Query → Open).



```
c:/euclid/goql/im/ap/o2dbase/thesis examples
11.gql
13.gql
14.gql
16.gql
17.gql
18.gql
20.gql
21.gql
22.gql
24.gql
```

Figure 11: The Open Query Dialog Box

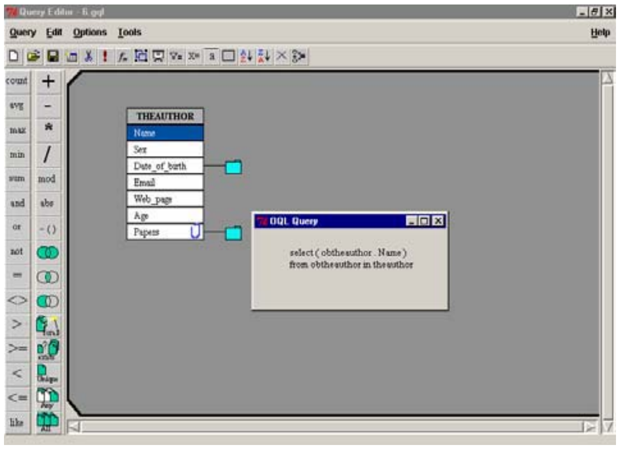
The Save Query button allows users to save a query in a file ('.gql' suffix) through the *Save Query* dialog box (Figure 12) (the path from the menu bar to the menu option is Query → Save and Query → Save As).



```
c:/euclid/goql/im/ap/o2dbase/thesis examples
11.gql
13.gql
14.gql
16.gql
17.gql
18.gql
20.gql
21.gql
22.gql
24.gql
```

Figure 12: The Save Query Dialog Box

The output of a graphical query is translated into o2 OQL and is presented in a special window the *OQL Query Output Window* (Figure 13).



```
select ( obtheauthor . Name )
from obtheauthor in theauthor
```

Figure 13: The *OQL Query Output Window*

5. Conclusions

This paper presented the system architecture of GOQL. In order to understand the system architecture, the UV and the FW interface of GOQL was also presented. The various components of GOQL and the way these components interact to generate the UV and the FW and to provide an ad hoc query construction and evaluation mechanism were discussed. Issues related to the implementation of the GOQL system architecture and the portability of GOQL across different DBMS platforms were also briefly discussed. Finally, GOQL's screen interface was presented. GOQL is a fully functioning graphical query language that is running on top of the o2 DBMS. Our current work involves continuous evaluation and maintenance of the system, involving correction of bugs and further enhancements of the system.

References

[1] Dix A., Finlay J., Abowd G. & Beale R., 1998. *Human Computer Interaction (Second Edition)*. Prentice Hall Europe.

[2] Keramopoulos, E., 2003, The GOQL Language, PhD Thesis, University of Westminster, 2003.

[3] R.G.G. Cattell & D.K. Barry (Eds.), *The Object Database Standard: ODMG 3.0* (Morgan Kaufmann Publishers, 2000).

[4] E. Keramopoulos, P. Pouyioutas & T. Ptohos, The GOQL Language, *submitted, International Journal of Visual Languages and Computing*, 2004.

[5] E. Keramopoulos, P. Pouyioutas & T. Ptohos, The System Architecture of the GOQL Language, Proc. *Intern. Conf. on Databases and Applications, Austria, 2004, 174-179.*

[6] E. Keramopoulos, P. Pouyioutas & T. Ptohos, The GOQL Graphical Query Language, *International Journal of Computers and Applications, 24*(3), 2002, 122-128.

[7] E. Keramopoulos, P. Pouyioutas & T. Ptohos, A Comparison Analysis of Graphical Models of Object-Oriented Databases and the GOQL Model, *Proc. 6th International Conference on Computers*, Crete, Greece, 2002, also in Recent Advances in Computers, Computing and Communications (WSEAS Series, 2002), 43-49.

[8] E. Keramopoulos, P. Pouyioutas & T. Ptohos, A Formal Definition of the Users View (UV) of the Graphical Object Query Language (GOQL), *Proc. International IEEE Conference on Information Visualization (IV'02)*, London, England, 2002, 211-216.

[9] E. Georgiadou & E. Keramopoulos, Measuring the Understandability of a Graphical Query Language through a Controlled Experiment, *Proc. BCS International Conference of Software Quality Management*, Loughborough, UK, 2001, 295-307.

[10] E. Keramopoulos, T. Ptohos & P. Pouyioutas, GOQL - A Graphical Query Language for Object-Oriented Databases, *Proc. IASTED AI2000 International Conference (Applied Informatics)*, Innsbruck, Austria, 2000, 129-133.

[11] E. Keramopoulos, P. Pouyioutas & T. Ptohos, The User's View Level of the GOQL Graphical Query Language, *Proc. International IEEE Conference on Information Visualisation (IV'99)*, London, UK, 1999, 81-86.

[12] E. Keramopoulos, P. Pouyioutas & C. Sadler, GOQL: a Graphical Query Language for Object-Oriented Database

Systems, *Proc. 3rd Basque International IEEE Workshop on Information Technology (BIWIT'97): Data Management Systems*, Biarritz, France, 1997, 35-45.

[13] o2 Technology, o2 User Manuals, 1995.

[14] J.K. Ousterhout, *Tcl and the Tk Toolkit* (Addison-Wesley Professional Computing Press, 1995).

[15] E.F. Johnson, Graphical Applications with Tcl & Tk (M&T Books, 1996)

APPENDIX I– The o2 ODMG Database Scheme of the Running Example of the Paper – Metadata Files

Person.o2
```
class Person inherit Object public type
    tuple( Date_of_birth: Date,
          Email: string,
          Web_page: string,
          Name: string,
          Sex: string)
method
    public Age: integer
end;
```

Author.o2
```
class Author inherit Person public type
    tuple(Papers: list(Paper))
end;
```

Date.o2
```
import schema o2kit class Date;
```

Document.o2
```
class Document inherit Object public type
    tuple(ISBN: string,
          Name: string,
          Publishers: Publisher,
          Price: real,
          Editors: set(Editor),
          Papers: set(Paper),
          Pages: integer)
end;
```

Names.o2
```
name TheAuthor :list(Author);
name TheDocument :set(Document);
name TheEditor :list(Editor);
name TheJournal :list(Journal);
name ThePaper :set(Paper);
name ThePerson :list(Person);
name Records98 :set(Proceedings);
name Records99 :set(Proceedings);
name ThePublisher :list(Publisher);
```

functions.o2
```
function display2;
function display3(name3: string): tuple(a3:string, b3:tuple(a31:integer, b31:string), c3:integer);
function display4(name4: string): Author;
function display5(name5: string, age: integer): tuple(a5:list(Author), b5:integer);
function display6(name6: string): set(Person);
function display7: string;
```

Proceedings.o2
```
class Proceedings inherit Document public type
    tuple ( Con_Date: tuple ( Start_Date : Date, End_Date : Date ),
            Place: tuple ( City : set(string), Country : string ) )
end;
```

Editor.o2
```
class Editor inherit Person public type
    tuple(Documents: list(Document))
end;
```

Paper.o2
```
class Paper inherit Object public type
    tuple(Title: string,
          First_page: integer,
          Last_page: integer,
          Authors: set(Author),
          Documents: set(Document),
          Keywords: set(string),
          References: set(Paper))
end;
```

journal.load.o2
```
#"c:\euclid\goql\im\ap\o2dbase\Date.o2"
#"c:\euclid\goql\im\ap\o2dbase\Person.o2"
#"c:\euclid\goql\im\ap\o2dbase\Paper.o2"
#"c:\euclid\goql\im\ap\o2dbase\Document.o2"
#"c:\euclid\goql\im\ap\o2dbase\Publisher.o2"
#"c:\euclid\goql\im\ap\o2dbase\Author.o2"
#"c:\euclid\goql\im\ap\o2dbase\Editor.o2"
#"c:\euclid\goql\im\ap\o2dbase\Proceedings.o2"
#"c:\euclid\goql\im\ap\o2dbase\Journal.o2"
#"c:\euclid\goql\im\ap\o2dbase\names.o2"
#"c:\euclid\goql\im\ap\o2dbase\functions.o2"
```

Journal.o2
```
class Journal inherit Document public type
    tuple(Volume: integer,
          Number: integer,
          Year: integer)
end;
```

Publisher.o2
```
class Publisher inherit Object public type
    tuple(Name: string,
          Address: string,
          Tel_no: string,
          Fax: string,
          Web_page: string,
          Documents: set(Document))
end;
```

APPENDIX II– The GOQL Data Structure Files for the Running Example of the Paper

Timestamp.goq
```
Year: integer
Month: integer
Day: integer
Hour: integer
Minute: integer
Second: integer
Millisecond: integer
```

Time.goq
```
Hour: integer
Minute: integer
Second: integer
Millisecond: integer
```

Interval.goq
```
Day: integer
Hour: integer
Minute: integer
Second: integer
Millisecond: integer
```

Date.goq
```
Year: integer
Month: integer
Day: integer
Day_of_year: integer
```

Hierarch.goq
```
Object Person
Object Paper
Object Document
Object Publisher
Person Author
Person Editor
Document Proceedings
Document Journal
```

Journal.goq
```
ISBN: string
Title: string
Publishers: Publisher
Price: float
Editors: set Editor
Papers: set Paper
Pages: integer
Year: integer
Volume: integer
Number: integer
```

Class.goq
```
Person
Paper
Document
Publisher
Author
Editor
Proceedings
Journal
Date
Interval
Time
Timestamp
```

Person.goq
```
Name: string
Sex: string
Date_of_birth: Date
Email: string
Web_page: string
Age: integer
```

Publisher.goq
```
Name: string
Address: string
Tel_no: string
Fax: string
Web_page: string
Publish: set Document
```

Editor.goq
```
Name: string
Sex: string
Date_of_birth: Date
Email: string
Web_page: string
Age: integer
Documents: list Document
```

Paper.goq
```
Title: string
Authors: set Author
Published_in: Document
First_page: integer
Last_page: integer
Keywords: set string
References: set Paper
Is_referenced: set Paper
```

Proceedings.goq_Place.goq
```
City : set string
Country : string  GOQLcomplexend
```

Proceedings.goqCon_Date.goq
```
City : set string
Country : string  GOQLcomplexend
```

display5.goq_b3.goq
```
a31: integer
b31: string GOQLcomplexend
```

display5.goq
```
a5: list Author
b5: integer GOQLcomplexend
```

Author.goq
```
Name: string
Sex: string
Date_of_birth: Date
Email: string
Web_page: string
Age: integer
Papers: list Paper
```

Proceedings.goq
```
ISBN: string
Title: string
Publishers: Publisher
Price: float
Editors: set Editor
Papers: set Paper
Pages: integer
Year: integer
Con_Date:  GOQLcomplex c:\euclid\goql\im\ap\o2dbase\journal\Proceedings.goq_Con_Date.goq
Place:  GOQLcomplex c:\euclid\goql\im\ap\o2dbase\journal\Proceedings.goq_Place.goq
```

Function.goq
```
display2
display3: GOQLcomplex c:\euclid\goql\im\ap\o2dbase\journal\display3.goq
          GOQL_parameter name3:string
display4: Author GOQL_parameter name4: string
display5: GOQLcomplex c:\euclid\goql\im\ap\o2dbase\journal\display5.goq
          GOQL_parameter name5:string age: integer
display6: set Person  GOQL_parameter name6: string
display7: string
```

display3.goq
```
a3: string
b3:   GOQLcomplex c:\euclid\goql\im\ap\o2dbase\journal\display3.goq_b3.goq
c3: integer GOQLcomplexend
```

Document.goq
```
ISBN: string
Title: string
Publishers: Publisher
Price: float
Editors: set Editor
Papers: set Paper
Pages: integer
Year: integer
```