Design considerations of the GOQL interface.

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

Design Considerations of the GOQL Interface

Euclides Keramopoulos
Department of Informatics, TEI of Thessaloniki,
P.O. Box 14561, Thessaloniki 54101, Greece
euclid@teithe.gr

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

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

Abstract. The Graphical Object Query Language (GOQL) is a query language that complies with the ODMG standard and which runs on top of the O_2 DBMS. The GOQL User Interface comprises the User's View (UV) and the Folders Window (FW). The UV is a graphical representation of an ODMG database scheme, which hides from end-users most of the perplexing details of the object-oriented database model. The FW is a condensed version of the UV that serves as canvas upon which ad-hoc queries are constructed. The paper addresses principles behind the design of User Interfaces and discusses features and characteristics of the GOQL User Interface.

Keywords Graphical Query Languages, Query Language, User Interfaces, OODBM

1. Introduction

The database languages evolution is strongly related partly to the evolution of user interfaces and partly to the evolution of database models and database systems in general. Until the early eighties not much attention was paid to the attractiveness, popularity and/or friendliness of user interfaces, mainly because users of these systems were highly trained and/or skilled professionals. As hardware costs plummeted and computer systems found their way into almost every aspect of life, the way computer systems were used also evolved. Nowadays, the majority of computer users need only to learn how to complete simple work tasks, whereas the problems they have to solve are usually expressed in non-computing terms. The change in the main type of user, from that of the highly skilled professional to that of the computer literate (unskilled or naive) user, meant that computers had to "acquire" or "compensate" for the skills that the new type of users lacked and to make interfacing with users simpler and friendlier. The advent of Graphical User Interfaces (GUIs), which utilise users cognitive skills and harness both advances in graphics technology and increased computing power, simplified and revolutionised the way users interface with computers and made computer systems accessibly to an even larger number of users. Nowadays, GUIs have become an essential part of any computer system and systems designers have come to accept that in order to improve users' productivity it is essential for a User Interface to address users' skills [1].

Herein we discuss issues related to the design of graphical query languages and the GOQL. In particular, we outline some of the principles and characteristics that influenced the design of graphical query languages; we examine some of these features in relation to the design of the User's View (UV) and the Folders Window (FW); we present the design of the GOQL, which is Graphical Query Language (GQL) that has the same expressive power as the OQL of the ODMG 3.0 standard [2] and which is the only GQL for the ODMG 3.0 that supports binding functions and method parameters. We conclude by illustrating aspects of GOQL's User Interface.

2. Design Principles and Characteristics of Graphical Query Languages

One of the first steps in our investigation into the design, definition and implementation of the GOQL was to identify and categorise principles and/or characteristics used in the design of GQLs. To do this we devised an analysis methodology that utilised elements of the approach taken in [4] for an analysis methodology on query languages and in [5] for the survey on graphical query languages on databases. We used our methodology to consider both languages and conceptual models that involve a graphical representation of data. The languages and conceptual models we considered using our analysis methodology include ODMG 3.0 [2], UML [6,7], COAD/YOURDON [8], ERM [9] and EERM [10], AMAZE [11], G-Log [12], GOMI [13], Khoshafian Model [14], Kaleidoquery [15], PICASSO [16], Pasta-3 [17], QBD* [18], SUPER [19], Gql [20], OdeView [21], QUIVER [22,23], MS-Access [24], Paradox [25], GQL [26], Business Objects [27] and Oracle [28].

Our findings/conclusions are summarised by the table in Fig. 1; a more detailed discussion on the methodology we employed and the characteristics/features, along with the languages and models we considered can be found in [3]

GOQL was designed based on the findings of the above investigation.

Features	User Type			Data Model				Understanding of modelled reality			Functionality			Language Form							Expressivity								Methods		Formal Definition				Human Factors		Output		
Graphical Query Languages	Naive	Skilled	Expert	Relational	Object-Oriented	Functional	Graphical Scheme	Top-Down	Browsing	Scheme Simplification	Retrieval	Update	Organisational	2D	3D	Textual	Form-based	Diagram-based	Icon-based	Hybrid	Predicates	Boolean & Set Operators	Arithmetic Expressions	Exist/Univ Quantifiers	Aggregate Functions	Group-by Operator	Binding Functions	Recursion	Yes	Not applicable	Syntactic Analysis	Semantic Analysis	Implementation	Evaluation	Colourful	Metaphors	2D	3D	Textual
PICASSO		✓		✓			✓				✓			✓				✓			✓	✓	✓	✓	✓	✓				✓			✓				✓		
Pasta-3		✓		✓			✓	✓			✓	✓	✓	✓				✓			✓	✓	✓	✓	✓	✓	✓	✓		✓			✓						✓
QBD*		✓		✓			✓			✓	✓			✓				✓			✓	✓	✓		✓	✓		✓		✓	✓	✓	✓	✓			✓		
G-Log			✓		✓		✓		✓		✓			✓				✓			✓	✓	✓	✓	✓	✓		✓			✓	✓	✓		✓				✓
AMAZE	✓				✓		✓				✓				✓			✓			✓	✓	✓		✓								✓		✓	✓		✓	
SUPER			✓	✓			✓	✓			✓	✓	✓	✓				✓			✓	✓	✓	✓	✓					✓			✓						✓
Ode View		✓			✓						✓	✓	✓			✓	✓				✓	✓	✓		✓				✓				✓				✓		
Gql	✓					✓	✓		✓		✓			✓				✓			✓	✓	✓	✓	✓	✓	✓			✓	✓	✓	✓						✓
GOMI		✓			✓		✓		✓		✓	✓	✓	✓						✓	✓	✓	✓	✓	✓			✓					✓				✓		
Kaleidoquery		✓			✓		✓		✓		✓				✓					✓	✓	✓	✓	✓	✓	✓			✓				✓	✓	✓	✓			✓
Quiver	✓				✓				✓		✓			✓			✓	✓			✓	✓	✓		✓				✓		✓	✓	✓	✓	✓				✓
Access		✓		✓			✓				✓	✓	✓	✓							✓	✓	✓	✓	✓	✓	✓			✓			✓				✓		

Figure 1. Features of Graphical Languages

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

GOQL was designed to address the needs of end-users and to provide an alternative graphical query language to OQL. Thus, GOQL was designed to comply fully with the features of the object model of the ODMG 3.0 [2] and its query language, OQL. 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. In this section we address the importance of graphical scheme representations and the use of metaphors in constructing these representations.

3.1. Graphical Scheme Representations

The importance of a graphical scheme representation of database constructs has been recognised in the late 70's following the proposal and success of the entity/relationship (E/R) model [9]. Since then graphical scheme representation has been used for the definition of data models (EERM [10]) and even for the representation of data in languages (UML [6,7]).

The main objective of graphical scheme representation is to provide a simple and user-friendly alternative to the way database structures are conceptualised. The complexity and level of detail in which graphical schemes represent characteristics of a database scheme reflect the technical competence of the indented target group of users. Thus, both skilled and expert users may find graphical schemes designed for naive users easy (and possibly frustrating) to use, whereas graphical schemes designed for expert users may require expertise and level understanding that skilled and/or naive users do not have.

We believe that most of the graphical schemes found in literature, with the exception of Kaleidoquery [15] and AMAZE [11], are addressed to expert users. This is because metaphors are not used in the graphical representation of these schemes. Moreover, all proposed schemes, represent graphically all the technical details of the underlying database model without trying to hide/metaphorically present some perplexing details that confuse non-expert users.

3.2. Desktop Metaphors

Quite frequently, in everyday life, attempts to explain or simplify something involve employing examples that the target audience can relate to. In [29] it is argued that an audience can relate better to the implications and complexity of something they are familiar with.

The use of metaphors is one of the most common teaching techniques, it is used to help children comprehend complex concepts. The same principle of using metaphors has also been used in computing as a way of making users, especially the naive ones, relate to and comprehend complex concepts. One of the most famous metaphors used in a software application is the turtle in LOGO [30], where the illustration of painting by following the movement of turtle's tail in the dust was very successful, especially with children, and it helped them learn and use LOGO more effectively. Similarly, metaphors have been used in database applications to make concepts of such applications conceptually simpler to users, especially to naive users.

In [1] it is claimed that the office environment is one of the most suitable fields to look for metaphors for database applications, because early database applications used to model and store business data that were stored as documents in folders. Moreover, according to [31], it is conceptually simpler for an office employee to work with a database interface that allows him/her to relate to pictures from his/her work environment (i.e. office). For example, [32] suggests that the use of a 'red book' as a metaphor will be successful in the interface design of an application that is developed for the employees of a company where a "real" red book is used for a specific purpose. However, the use of metaphors requires careful consideration, as choosing the wrong metaphor can easily confuse users and lead them to misinterpret the intended semantics with possibly disastrous results.

In conclusion, a wisely selected/used metaphor that a target audience can relate to, can conceptually simplify complex concepts.

3.3. Graphical Scheme Design

A graphical scheme is a representation of structures of the underlying database model. Designers of such schemes maintain a one-to-one correspondence between elements of the graphical scheme and the corresponding constructs of the underlying database scheme. An immediate implication is that the graphical scheme representation is constrained by the constructs of the underlying database model and it could not represent the database scheme of another database model. If the graphical scheme is to be independent of the underlying database model, the set of metaphors used must be independent of the structures of the underlying database model. For example, in User's View the metaphor folder can be interpreted either as a class object or as an entity. The set of metaphors that a graphical scheme supports must cover all the different features of the scheme's underlying database model. However, a subsumption relationship can be defined between certain database models; this leads us to believe that a graphical scheme for a database model that subsumes a number of other models will also cover the features of all the data model that are subsumed by the database model for which the graphical scheme was created. In particular, the OODBM is a database model whose features subsume the cfeatures of the relational, the nested relational, the complex objects and the semantic database models [33]. Thus, a graphical scheme which can represent the features of the OODBM can also represent the features of all the models it subsumes. The User's View is designed for the OODBM, in this way that it can represent constructs of all the database models that OODBM subsumes.

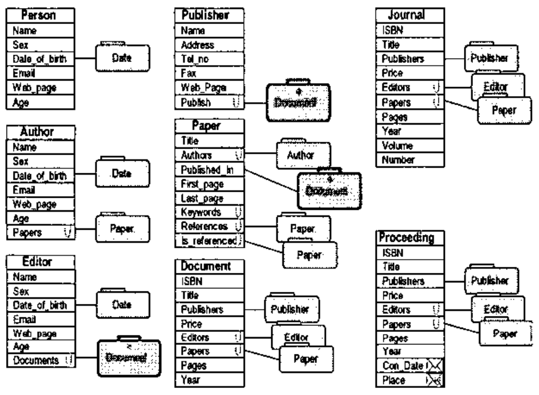

Figure 2. The User's View

3.4. The User's View of GOQL

Graphical schemes were defined to be faithful and precise representations of an underlying database scheme. However, the representation of all possible details of an underlying scheme can be overwhelming for some users, so it may be preferable for certain details to be hidden from certain classes of users. Hiding the underlying scheme's perplexing details combined with the use of appropriate metaphors can simplify the graphical scheme and make its use more effective for naive users. The User's View is such a graphical scheme. In Fig. 2 the User's View for a publisher's database is given.

3.5. GOQL and the Problem of Complex Databases – The Condensed View

GOQL deals with the problem of representing a complex database scheme by adopting a hybrid approach, which involves the top-down approach, the browsing approach, and the scheme simplification approach.

Fig. 3 presents the Folders Window (FW), which is a ***condensed*** graphical representation of the scheme UV. It consists of a number of closed folders, one for each of the defined classes/entities. Users can choose to 'open' any of the included folders to examine the features of that class. By selecting a folder the graphical scheme is moving one level down (top down approach) for the particular class/entity. At this level relationship browsing can allow a user to open any of the contained relationships, whereas using selecting elements of these relationships can allow a user to develop/navigate part of the scheme that is of his/her interest; in other words a scheme simplification is achieved by scheme developing, Fig. 4. We believe that this combination of the three approaches provides a straightforward mechanism to browse schemes of even complex database schemes. Users can also use this approach to construct subschemes of the initial scheme that meet query requirements.

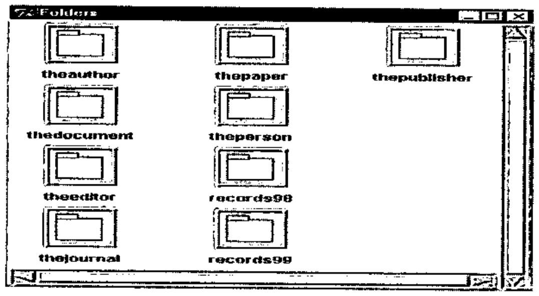

Figure 3. The Folders Window

4. GOQL Design Considerations

When designing the GOQL, we had as a target the implementation of a graphical query language, which would support the whole repertoire of the OQL of ODMG 3.0 [2]. In addition, we wanted to present expert users with a language that they could use more productively than other GQLs or the OQL, at the same time we wanted to present naive users with a language that they would be able to use with the least possible training. To achieve these targets we used metaphors for the graphical representation of the scheme and we tried to give a visual look to the query construction that will not trouble user with perplexing symbols and diagrams and which will highlight important elements of a query, using colour and special symbols, which attract the attention of the user.

4.1. Visual Representations

In [5] eleven (11) different types of visual representations were identified as being used in various graphical query languages. The three more important and commonly used visual representations are the form based one, the diagram based and the icon-based representation. A combination of two or more types of representation results in hybrid systems. According to [5], the combinations adopted/used so far are: a) forms and diagrams, b) diagrams and icons and c) forms, diagrams and icons. Overall, hybrid systems produce better query representations because designers can choose the features of each representation that will create a better and more productive result. Furthermore, quite frequently users are used to specific pictures from their life, which are presented by different types of presentation; thus, it is very difficult to illustrate these pictures using only one type of representation. The above led us to use forms for the presentation of data, diagrams for the presentation of operators and icons for the representation of toolbars, i.e. the GOQL interface is a hybrid one

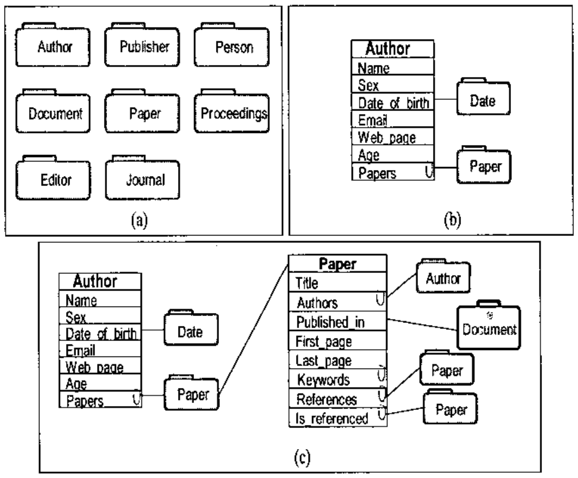

Figure 4. The Folders Window, Scheme Developing, and Relationship Browsing

4.2. Cognitive and Technical Aspects

GQLs are focused on specific classes of users and this is reflected by their design. GOQL is aimed at all types of users; thus, the GOQL's design had to incorporate both cognitive and technical characteristics that address the needs of each class of users. Among the cognitive targets of GOQL's design was to create an easy to learn language with natural language characteristics, which will be easily used. To achieve this, a number of desktop metaphors were used, especially for the representation of the graphical scheme. For the representation of the various operators simple shapes have been selected having as a criterion for the selection the type of the operator (unary or binary) and the number of its operands. The overall aim was to allow users to intuitively recognise the semantics of the various operators and use them accordingly.

Finally, in order to enhance the 'readability' of the graphical queries, colour was used. For the technical aspects of the design we developed a language with a hybrid query construction mechanism. To improve this mechanism we incorporated in the design the majority of the known methods of formulating a query. Moreover, all of the GOQL's tools are visual formalisms, i.e. tools, that are formally defined to be used by a computer and which also can be visualised by users [34].

4.3. Shape

According to [35] the shape of elements of a language can be utilised as an effective way of differentiating between these elements. In GOQL, we tried to utilise the above idea to differentiate between operators. Thus, operators were categorised and different shapes were introduced to represent each of the categories. Operators within each category are identified by the category's symbol and a word that identifies an operator. The shapes used are:

- Hexagons: used to represent boolean operators.
- Small ovals: used to represent unary operators.
- Large ovals: used to represent binary operators.
- Circles: used to represent sorting.

4.4. Colour

In devising a strategy for the use of colour in GOQL, we followed the suggestions of [36, 1]. In particular, the monitor was adjusted to display only shades of grey to check whether the used colours, can be easily read by the majority of users and only seven (7) colours were used to represent all the GOQL's features. Moreover, highly contrasting colours that be easily recognisable / readable in a monochrome monitor were chosen.

Furthermore, we utilised a colour convention that we felt was intuitive as it is also used in a similar way in other areas such as a traffic/work environment [36, 1]. In particular, red was used for alerts, for example when function parameters have to be provided. Green was used to indicate that all is clear, i.e. no syntax error. Yellow was used as a sign of caution; the draw attention action is given by painted yellow semicircles, which indicate where a condition has been inserted. Dark blue was used to highlight selected items; in particular we chose to use a dark colour to highlight projected items and the blue colour to achieve the differentiation for the projected items. We used different colours to present each of the metaphors. The colours used, which also give a nice result in a grey scale, are:

- For the folder, the turquoise green.
- For the briefcase, the brown.
- For the envelope, a mix of red and orange.
- For the clip, the light blue.

For the background a shade of a grey was used, which is a neutral colour, i.e. it does not make colours painted on it to look darker or lighter; it is friendly and unobtrusive [35]. Finally, a thin black border is used with each of the defined tool-shapes of GOQL to make them clearly recognisable by all users.

5. Conclusions

The paper presented the GOQL language and addressed the advantages of our language as compared to other GQLs Amongst the most important advantages of GOQL is the database model independence that it supports and its simplified user interface that hides any perplexing details of the underlying model(s). The paper discussed the principles and characteristics of graphical query language and showed how these were incorporated in the design and development of GOQL. The language's user interface, namely the User View was presented. GOQL is fully functioning and is running on top of the O_2 DBMS. Our current work involves continuous evaluation and maintenance of the system, involving correction of bugs and further enhancements of the system.

6. References

[1] Dix A, Finlay J, Abowd G, Beale R. Human Computer Interaction, 2nd Edition. Europe: Prentice Hall; 1998.

[2] Cattell R G G, Barry D K editors. The Object Database Standard: ODMG 3.0. Morgan Kaufmann Publishers; 2000.

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

[4] McDonald N H & McNally J P. Query Language Feature Analysis by Usability. Computer Languages 1982, 7, 103-24.

[5] Catarci T, Costabile M F, Levialdi S, Batini C. Visual Query Systems for Databases: A Survey. Journal of Visual Languages and Computing 1997; 8(2); 215 – 60.

[6] Booch G, Rumbaugh J, Jacobson I. The Unified Modeling Language User Guide. Addison – Wesley Object Technology Series; 1998.

[7] Gornik D. UML for Data Modeling Profile. Rational Software Whitepapers; TP162; 2002 www-306.ibm.com/software/rational/library/whitepapers/tp162.html [23/01/04].

[8] Coad P, Yourdon E. Object-Oriented Analysis, 2nd Edition. Prentice Hall; 1991.

[9] Chen P P. The Entity-Relationship Model: toward a Unified View of Data. Journal of ACM Transactions on Database Systems 1976; 1(1); 166 – 92.

[10] Schiffer G, Scheuermann. Multiple Views and Abstractions with an Extended-Entity Relationship Model. Journal of Computer Languages 1979; 4; 139 – 54.

[11] Boyle J, Leishman S, Gray M D. From WIMPS to 3D: The Development of AMAZE. Journal of Visual Languages and Computing 1996; 7(3); 291 – 319.

[12] Paredaens J, Peelman P, Tanca L. G-Log: a Graph-based Query Language. Journal of IEEE Transactions on Knowledge and Data Engineering 1995; 7(3); 436 – 53.

[13] Jun Y S, Yoo S I. GOMI: A Graphical User Interface for Object-Oriented Databases. Proceedings Int'l Conference on Object-Oriented Interface Systems; 1995; 238 – 51.

[14] Khoshafian S. Object-Oriented Databases. John Wiley & Sons Inc; 1995.

[15] Murray N, Paton N, Goble C. Kaleidoquery: A Visual Query Language for Object Databases. Proceedings 4th IFIP Working Conference on Visual Database Systems; 1998 May 27–29; L'Aquila, Italy; 247 – 57.

[16] Kim H, Korth H F, Silverschatz A.: PICASSO: A Graphical Query Language. Software-Practice and Experience 1988; 18(3); 169 – 203.

[17] Kuntz M, Melchert R. Pasta-3's Graphical Query Language: Direct Manipulation, Co-operative Queries, Full Expressive Power. Proceedings 15th Int'l Conference on Very Large Databases; 1989; 97 – 105.

[18] Angelaccio M, Catarci T, Santucci G. QBD*: A Graphical Query Language with Recursion. IEEE Transaction on Software Engineering 1990; 16(10); 1150 – 63.

[19] Dennebouy Y, Andersson M, Auddino A, Dupont Y, Fontana E, Gentile M, Spaccapietra S. SUPER: Visual Interfaces for Object + Relationship Data Models. Journal of Visual Languages and Computing 1995; 6(1); 73 – 99.

[20] Papantonakis A. Gql, a Declarative Graphical Query Language Based on the Functional Data Model. PhD Thesis, Birkbeck College, University of London; 1995.

[21] Dar S, Gehani N H, Jagadish H V, Srinivasan J. Queries in an Object-Oriented Graphical Interface. Journal of Visual Languages & Computing 1995; 6(1); 27–52.

[22] Chavda M, Wood P T. Combining Constraints and Data-Flow in A Visual Query Language. Proceedings IEEE Symposium on Visual Languages; 1997 Sep 23 – 26; Capri, Italy; 125 – 6.

[23] Chavda M, Wood PT. Towards an ODMG-Compliant Visual Object Query Language. Proceedings 23rd VLDB Conference; 1997; Athens, Greece; 456 - 65.

[24] Microsoft Corporation Ltd. Microsoft Windows 2000 Professional Resource Kit. Microsoft Press; 2000.

[25] Borland. Paradox for Windows; 1995.

[26] SOFT TOOLRACK Ltd. GQL (Graphical Query Language); 1995.

[27] Business Objects Ltd, 2003, Available at:http://www.businessobjects.com/products/query_report_analysis.htm

[28] Oracle Corp. Simple Strategies for Complex Data: Oracle9i Object-Relational Technology. An Oracle Technical White Paper; 2002. otn.oracle.com/products/database/application development/pdf/simple_strat_for_complex_rel2.pdf [23/01/2004].

[29] Torgny O. Metaphor – a Working Concept. Proceedings Contextual Design – Design in context; 1997; Stockholm, Sweden; 3 – 14.

[30] Slack J. Turbo Pascal with Turtle Graphics. West Publishing Company; 1990.

[31] Collins D. Designing Object-Oriented User Interfaces. Benjamin/Cummings Publishing Company Inc; 1995.

[32] Lovgren J. How to choose good metaphors. Journal of IEEE Software 1994; 11(3); 86–8.

[33] Pouyioutas P. Formalising the Extended Object-Oriented Database Model. PhD Thesis, Birkbeck College, University of London; 1996

[34] Harel, D. On visual Formalism. Communication of the ACM 1988; 31(5); 514 – 30.

[35] Foley J, Van Dam A. Computer Graphics Principles and Practice, 2nd Edition. Addison – Wesley Publishing Company; 1990.

[36] Newman W M, Lamming M G. Interactive System Design. Addison – Wesley Publishing Company; 1995.