QBI: An Iconic Query System for Inexpert Users

Antonio Massari, Stefano Pavani, Lorenzo Saladini

Dipartimento di Informatica e Sistemistica Universita' di Roma "La Sapienza"
Via Salaria, 113 - 00198 Roma, Italy
massari@infokit.dis.uniroma1.it

Abstract

We present a general purpose query interface for inexpert users based on the manipulation of icons. The user perceives the reality of interest as structured in classes and attributes while the system internally maintains a schema rich of semantic information. The query language, fully visual, is based on the select and project paradigm that has been proven to be easy to understand. No path specification is required for composing a query. Automatic feedbacks based on natural language generation and cardinality constraints analisys help the user in specifying her/his requests.

1 Introduction

QBI (Query By Icon) is a system that allows users to query and understand the information content of a database by manipulating icons. It is especially addressed to the category of users characterized by the contrasting peculiarities of having limited skill and unpredictable exigences in their requests.

Even though QBI is a general purpose query interface, it takes its origins as interface for a distributed database of radiological images [1]. In this environment, the high response times due both to large data sizes and to limited network speeds, made not feasible a querying approach based on the database instances browsing [2]. As a consequence, in QBI the query specification phase has made clearly distinct from the result analysis phase. All the feedbacks associated to the user actions are based on the elaboration of intensional information, since it is supposed to be invariant in time and not large in size.

The system maintains an internal schema of the database structure made according to a semantic model, the Graph Model. This schema, not visible to the user, is a labeled graph representing both the structural information (e.g. classes and relationships) and consistency constraints of the database. The external view is based on the concept of Generalised Attribute by means of which the database appears structured in a flat way. More specifically, the user perceives the existence of classes of objects and a set of properties, called Generalised Attributes, for each class.

As a result, even complex queries can be expressed using the *select-project* paradigm that has be proven to be very simple to understand [3]. Finally the possibility of reusing queries in the specification of more complex ones, further improves the usability of the system.

QBI takes advantage of iconic metaphors for the visualization of both structural information and constraints. The implicit ambiguity of iconic representation is resolved by using automatic generated natural language.

2 Basic Choices

Semantic and object oriented models introduce conceptually rich constructs for the definition of the database schema [4]. Users interacting with query languages based on these models, however, encounter two sources of difficulty:

- understanding the schema and
- specifying logical access paths.

As a matter of facts, the mental effort required to inexpert users for understanding the meaning of semantic constructs, such as cardinality constraints or complex relationships, represents a bottleneck for an effective use of the query language. Moreover, the visual representation of such systems is typically based on diagrams [5], often resulting too complex to be accepted by inexpert users.

Studies have shown how queries involving the navigation on the database schema are more difficult to express and often expressed incorrectly [3]. Navigation in semantic data models is simpified by the use of *implicit joins*, nevertheless the user must still specify the appropriate connection in her/his query.

The design of QBI has been made taking into account the previous issues. The following guidelines have been carried out:

- Presentation to the user of a simplified structure of the reality of interest.
- Logical path independence in query specification.
- Possibility of incremental composition of queries by defining *views*.

2.1 The Generalised Attribute

Due to their natural simplicity, the external representation of the database structure our query interface offers is based

exclusively on the concepts of *class of objects* and *attribute of classes*. The user perceives the reality of interest as a set of classes each one having several properties called *Generalised Attributes* (GA in the following). As attributes in the ER model represent elementary properties of entities, a GA expresses a *generic property* of a class; for example the set of `cars` owned by a `person` is treated as an Generalized Attribute of `person` as well as its `name` or `birthdate`.

In our system the semantic model used for defining the internal representation is the Graph Model [6]. The major components of this model are: the class of object, the relationship among classes, and the ISA relationship between a class and its superclass. Along with the previous basic concepts it is possible to express cardinality constraints for the participation of class instances into the relationships. A Graph Model schema can be expressed in term of a labeled graph called Typed Graph. The nodes of a Typed Graph represent both classes and relationships. More specifically, there are three different kind of nodes: the *class nodes*, that can be *printable* or *non printable* depending on whether they represent domains of values or abstract classes, and the *role nodes* representing relationships among an arbitrary number of classes. Edges represent the connections between class nodes and a role nodes.

The GA is strictly related to the concept of *path* on a Typed Graph, where a path is an alternate sequence of adjacent class and role nodes always starting and ending with a class node. Given two class nodes n_c (*picked class node*) and $\overline{n_c}$, a path p starting in n_c and ending in $\overline{n_c}$ defines a GA of n_c as a multivalued function mapping each instance x of n_c in a set of instances of $\overline{n_c}$ [1].

A GA can be either *single valued* or *multivalued* according to the cardinality constraints of the role nodes involved in the path. The type of the last node of the path determines the *type* of the associated GA.

The concept of GA is thus part of the external representation; it represents the way by which the user perceives the relationships among concepts. The system is responsible for the generation of the GAs, in paricular the GAs of a class node are automatically calculated by generating all the paths starting from the class node. Is worth noting how, by means of the GAs of whatever class node, the user can observe the entire structure of the database. For example a GA of the class node `person` can be `"All the hospital located in the same city where the person lives"`. By observing such an attribute, the user perceives that a `hospital` is located in a `city`, that is a part of the schema not directly connected to `person`. The same information, of course, could be got more easlily by observing the GAs of `hospital` and `city`.

If no limitation on the path structure were imposed, the set of GAs of a class node wolud result infinite. As a consequence, it is necessary to define a criteria for considering a finite subset of GAs.

Our idea is to define a *semantic distance function* returning, for each GA, a value representing the meaningfulness of the GA with respect to the picked class node. Many aspects of the GA structure can be considered for the definition of semantic distance, for example the path length or the cardinality constraints. The finite set of GAs the system shows to the user consists of all those GAs "meaningful enough" to be reasonably used for querying.

[1] The formal definition of the function has not been included for brevity.

2.2 Query Specification

The query language is based on the select-project paradigm: a query is expressed by first defining the conditions that determine a subset of the picked class node (selection) and then specifying those GAs that are going to be part of the output result (projection). In both the selection ad projection phases no path specification is required to the user.

Each query, considered only in its selection part, can be treated as a subset of the picked class node; as a consequence it can be internally represented as a new derivate class node connected by means of an ISA relationship to the picked class node.

Consider, for example, a medical database dealing with patients and diseases. Suppose there are, among the others, the class nodes `Patient`, `Disease` and a subclass of this node `Heart disease`. A multivalued GA of the class node `Patient` will be `"All the diseases the patient has had"`. Suppose the user is interested in all the patients that have had heart diseases; the selection part of this query will be composed by the atom `"All the diseases the patient has had include some heart disease"` obtained by comparing the above GA and the constant set `Heart disease` with the connective `not-disjoint`. Note that the class node `Heart disease` has been used as a comparative constant in the atomic condition.

The incremental composition of a query is performed by reusing a derivate class node as a constant in the specification of a query.

3 Visualization

The visualization of classes and GAs is based on icons and automatic generated natural language.

Icons represent both:

- primitive and derivate classes. The icon represents all the objects belonging to the class.
- the set of objects related to a generic instance of a class via a GA. More specifically, let γp be a GA of a class node n_c and p the associated path starting in n_c and ending in $\overline{n_c}$. The set of objects, belonging to $\overline{n_c}$, and related to each instance x of n_c via γp, is visualized as an icon.

An icon results in the composition of several *graphical items* each one bearing a specific meaning. The *picture* is the most important graphical item, since it sintesyzes the meaning of the type of objects being represented. Other graphical items are used for presenting cardinality and type compatibility constraints.

Natural language plays an essential role for disambiguating the meaning of both GAs and derived classes. Using the techniques described in [7] it is possible to automatically generate sentences describing GAs and derived classes.

4 The Prototype

A prototype of QBI has been developed for the MS-Windows environment using the toolkit XVT [2] and the C language. The functionalities currently available allow the specification of both the selection and projection phase of a query, as well as the management of the GA sets. A query can be saved as new derived subclass and used, as comparative constant, in the specification of a more complex one, thus allowing the user to adopt the incremental composition querying approach.

4.1 Selecting GAs: Metaquery

The prototype includes a facility for browsing the GA set of a class. By means of this tool the user can locate the GAs of the picked class node she/he looking for and use them for querying. The set of GAs is sorted by semantic distance with respect to the picked class node: in this way the most meaningful GAs are shown first.

The manual browsing of the GA set can be a not trivial task if the user is interested in GAs semantically far from the picked class node. In order to address this problem, the browsing facility permits the specification of filter conditions on the GA set. For example, it is possible to select only those GAs of a certain type or only the single valued ones. A complex filter condition can be expressed using different operators; since the operation of GAs selection by filtering can be regarded as a query on the schema of the database, it has been called *metaquery* operation.

4.2 Interface Description

Three windows compose the QBI interface: the Workspace, the Query and the Browser window.

Workspace Window. This window is the container of both the primitive and derivate classes; each icon in this space corresponds to a class node of the underlying Typed Graph. The user can freely arrange the icons in the workspace and create duplicates. Pointing an icon corresponds to make the associated class node the picked class node.

Query Window. The query window is activated by double clicking an icon in the Workspace. By using this window it is possible to specify both the selection conditions and the projection of a query, as well as to read its natural language description. In the Query window are by default present those GAs semantically very close to the picked class node. This set of GAs (*initial GA set*) is determined by a constant threshold value for the semantic distance.

Browser Window. If the initial GA set does not contain those GAs that are necessary to the user for composing the query, it is possible to look for more attributes by activating the appropriate command in the Query Window. The window dedicated to this purpose is the Browser window. In the Browser window there is a scrolling list of GAs as well as several spaces and buttons for the metaquery specification. Each item in the list corresponds to an icon representing the GA and a natural language sentence describing it. The user can drag these icons from the Browser window and drop them in the the Query window for either composing a selection condition or specifying the output result part of the query. The metaquery operators allows the user to restrict the search of the desired GAs within a smaller set. Among the various metaquery operators, it is possible to select all the GAs having a certain type, being single valued or "dealing with" a certain concept.

[2] This toolkit guarantees the portability of the application in many window systems

5 Conclusions and future work

We have presented a general purpose query system based on icons for which the usability is the main target. Informal tests seem to shown the high value of this interface in terms of friendliness both for the schema understanding and query formulation phases.

Beyond the addition of useful features like transitive closure and aggregative operators, the major effort must be done in the direction of proving the usability of QBI with respect to other systems in a scientific manner. For this reason the work done so far has to be considered as a first step.

We believe the most significative comparative analisys has to be made with respect to other visual interfaces for which usability results, with respect to non graphical languages, have already been achieved. In our opinion QBD [5] is the best candidate for such a comparison, since it is addressed to the same category of users and actual installations are currently available.

References

[1] S.K. Chang, T.Y Hou and A. Hsu. Smart Image Design for Large Image Databases. In *Journal of Visual Languages and Computing*, Vol. 3 No.4, December 1992

[2] A. Motro, A.D' Atri and L. Tarantino. KIVIEW: The Design of an Object Oriented Browser. In *Proc. 2nd Conf. on Expert Database Systems, Virginia USA*, pp 17-31, 1988.

[3] D. Greenblatt and J. Waxman. A study of three database query languages *Database: Improving Usability and Responsiveness , B.Scheneiderman, ED. New York:Academic.*, 1978.

[4] R.B. Hull and R. King. Semantic Database Modeling:Survey, applications and ressearch issues. In *ACM Computing Surveys*, Vol. 19 No.3, 1987

[5] M. Angelaccio, T. Catarci and G. Santucci. QBD*: A Graphical Query Language with Recursion. In *IEEE Transactions on Software Engineering*, Vol. 16, No. 10, 1990.

[6] T. Catarci and G. Santucci. Fundamental Graphical Primitives for Visual Query Languages. In *Information Systems* , Vol. 18, No. 3, 1993.

[7] G. Bono and P. Ficorilli. Natural Language Restatement of Queries Expressed in a Graphical Language. In *Eleventh International Conference on Entity-Relationship Approach*, Germany, North-Holland, 1992