Visual Query of Completely Encapsulated Objects

Antonio Massari
Dipart. di Informatica e Sistemistica
University of Rome "La Sapienza"
00198 - Roma, Italy

Panos K. Chrysanthis
Dept. of Computer Science
University of Pittsburgh
Pittsburgh, PA 15260, U.S.A.

Abstract

This paper presents a general purpose object query system and language, called QBI (Query by Icon), that involves the manipulation of icons for composing a query. It requires neither special knowledge of the content of the underlying database or understanding of the details of the database schema, hence it is equally useful to both unsophisticated and expert users. A user perceives the database as classes of objects and generalized attributes while the system internally maintains a schema of the underlying database rich in semantic information. No relationship specification is required for composing a query. Furthermore, automatic natural language feedback and cardinality constraints analysis assist users in query specification.

1 Introduction

Databases are being incorporated in more and more complex applications such CAD/CASE/CIM systems, multimedia systems and heterogeneous information systems. Semantic and object-oriented (OO) models have been proposed in order to deal with the inherent limitations of the traditional data models as well as the complexity of the structure of these, typically distributed, advanced application databases [12]. In addition, these models open new possibilities in query processing based on fifth generation visual query languages [3].

Semantic and OO models provide few, conceptually rich, constructs for defining the schema of a database. Typically, semantic and OO models result in a database schema which can be expressed and visualized as a type of graph. To capture the various semantic aspects of a model, such graphs are composed of different kinds of nodes and edges enhanced with labels and annotations for expressing constraints. A query is expressed by defining a *logical access path* on the graph representing the database schema. The allowable paths and the meaning of a specific path, i.e., the way that objects along a path are related, depend on the model and the query language itself.

The effort required by users, particularly unsophisticated ones, for understanding the meaning of various constraints and complex relationships, becomes a stumbling block for an effective use of such query language. Although navigation on semantic and OO database schema is simplified by the use of *references (implicit joins)*, the user must still specify the appropriate connections. Similarly, in the cases where a database schema is visualized as a diagram [2], and queries are formulated by drawing nodes and edges to be matched in the schema diagram, diagrams often become too complex for unsophisticated users. Particularly difficult is the visual formulation of queries involving cyclic multiple paths.

In this paper we present QBI (Query By Icon), a general purpose query system developed to support the exploration and query of large distributed databases. QBI is based on the notion of *complete objects*, i.e., completely encapsulated objects, that provides logical access path independence and incremental composition. Hence, it requires no special knowledge of the content of the underlying database nor understanding of the details of the database schema.

The complete object model is motivated by the *universal relation* approach which is intended to provide the user with a simplified model in which she/he can compose queries without considering the underlying structure of the relations in the database [8]. Universal Relation systems, assume that the database is structured as a single relation whose attributes encapsulate all the semantics of the underlying database. Similarly, by fully encapsulating both implicit and explicit relationships among objects within each object, the complete object model allows the whole database to be seen from within any single object. More specifically, a user perceives the existence of classes of objects and a set of properties for each class, called the *generalized attributes*. Thus, in QBI, each object provides a view of the whole underlying database from the point of the object and can be thought of as an independent "universal relation" or complete object.

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 automatically generated natural language. In general, the main difference between QBI and the other iconic interfaces proposed in the literature [7, 11, 10] is in the way icons are defined and used for expressing concepts. As opposed to QBI, other systems do not usually assign uniform semantics to icons and adopt instance browsing as the principal querying strategy.

QBI internally uses the *Binary Graph Model* (BGM), a semantic data model, for capturing most of the aspects of the underlying database. In the next section, BGM is formally defined. In Section 3, we introduce the Complete Object Model and formalize it in terms of BGM. Section 4 describes the salient features and components of QBI prototype whereas Section 5 illustrates the functionality of the prototype through its use to query a radiological database.

2 The Binary Graph Model

The *Binary Graph Model* is central to QBI providing the mathematical representation for the Complete Object Model and being employed as the internal representation of QBI. BGM is derived from the Graph Model [5] that was proposed to support multiparadigm interfaces.

The major components of BGM are: the class of objects, the binary relationship among classes, the ISA relationship between a subclass and its superclass, and cardinality constraints for the participation of class instances into relationships. A BGM schema can be expressed as a labeled graph called *typed-graph.*

DEFINITION 2.1: [Typed Graph] A ***typed-graph*** $g(N, E)$ is a labeled multigraph. The set N of nodes consists of ***class nodes*** N_C representing classes of objects and ***role nodes*** N_R representing relationships between two classes. Class nodes can be either ***printable*** or ***nonprintable*** depending on whether they represent domains of values or abstract classes. An edge in E can only link a class node to a role node and is associated with a unique label in L. Each role node has a degree equal to two.

A class node is said to be ***adjacent*** to a role node if there is an edge connecting the two nodes. Each role node will have exactly two adjacent class nodes. When the adjacent class nodes are coincident we say that the role node is ***reflexive.*** In this case labels on edges are useful for disambiguating the two edges.

A BGM database is defined as a triple $< g, c, m >$, where g is a typed-graph, c is a set of constraints, and m is an interpretation. The schema of a database is represented by g and c whereas an instance of a database (extension) are represented by the notion of interpretation.

DEFINITION 2.2: [Interpretation] Let g be a typed-graph. An ***interpretation*** for g is a function m mapping each class node $n_c \in N_C$ to a set $m(n_c)$ of objects and each role node $n_r \in N_R$ to a set $m(n_r)$ of pairs $< lbl_1(n_r) : x_1, lbl_2(n_r) : x_2 >$, where lbl_1, lbl_2 are functions returning the labels of the two edges connected to n_r (lbl_1, lbl_2: $N_R \rightarrow L$) and $< x_1, x_2 > \in m(n_{c1}) \times m(n_{c2})$ where n_{c1} and n_{c2} are the adjacent class node of n_r.

That is, an interpretation specifies the valid combinations of values from the underlying classes.

DEFINITION 2.3: [Constraints] The set of constraints c on a database is expressed by means of a constraint language. In this paper we will refer to the following three constraints:

- $ATLEAST(k, n_{c1}, n_r)$ (minimum cardinality constraint) specifies that an instance of class node n_{c1} can participate in at least k interpretation involving the adjacent role node n_r.

- $ATMOST(k, n_{c1}, n_r)$ (maximum cardinality constraint) specifies that an instance of class node n_{c1} can participate in at most k interpretation involving the adjacent role node n_r.

- $ISA(n_{\hat{c}}, n_c)$ (subclass-superclass relationship) specifies that the class $n_{\hat{c}}$ is a subset of the class $n_{\hat{c}}$ (i.e.,$m(n_{\hat{c}}) \subseteq m(n_c)$). The role nodes connected to n_c are considered as also being connected to $n_{\hat{c}}$, i.e., $n_{\hat{c}}$ ***inherits*** the edges of n_c. Currently we assume single inheritance, hence each class node has to belong to one and only one ***class hierarchy.***

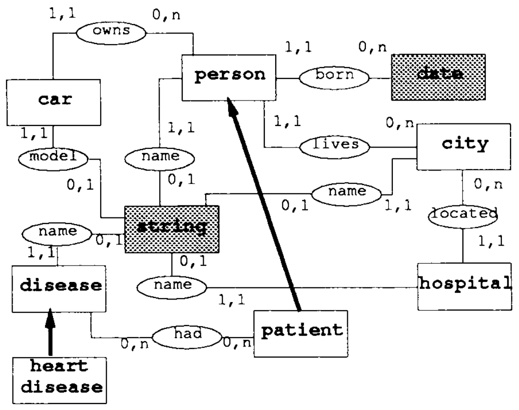


Figure 1: A Typed Graph

In order to facilitate type checking of query expressions, we define the notion of ***type*** of a class as a class hierarchy. That is, the class nodes belonging to a class hierarchy have the same type. Note that if n_{c1} and n_{c2} have different types, their interpretations are disjoint.

Figure 1 shows an example of a typed graph. The rectangular boxes represent class nodes (the printable ones are grayed) while the ovals represent role nodes. No label on an edge is shown since there are no reflexive role nodes that need to be disambiguated. The annotations (m,n) on edges represent (ATLEAST,ATMOST) cardinality constraints. ISA constraints are denoted by a thick arrow from a subclass node to its superclass.

3 The Complete Object Model

In QBI, the concepts of ***class of objects*** and ***attribute of a class*** exclusively form the external representation of the database structure due to their natural simplicity. Specifically, a user perceives the underlying database as a set of classes, each having several properties called ***Generalized Attributes*** (***GA***). In the same way that attributes in the ER model [6] 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 a GA of person

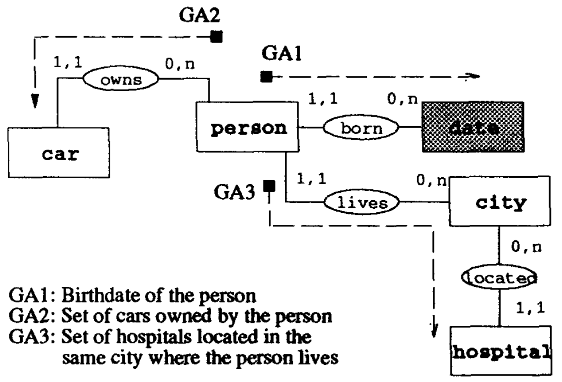

GA2
GA1
1,1
0,n
owns
car
person
1,1
0,n
born
date
GA3
1,1
0,n
lives
city
0,n
located
1,1
hospital
GA1: Birthdate of the person
GA2: Set of cars owned by the person
GA3: Set of hospitals located in the same city where the person lives

Figure 2: Examples of Generalized Attributes

in a way similar to that of simple attributes such as its name or birthdate.

A GA is strictly related to the concept of *path* in a typed-graph, where a path is a sequence of adjacent class and role nodes always starting and ending with a class node.

DEFINITION 3.1: [Path] Let $\mathcal{G}$ be a typed-graph. A ***step*** s on $\mathcal{G}$ is the triple $<class_1(s)$, $role(s)$, $class_2(s)>$ where $class_1(s) = n_{c1}, class_2(s) = n_{c2} \in N_C$ are adjacent to $role(s) = n_r \in N_R$. A ***path*** p on $\mathcal{G}$ is the sequence $s_1, s_2, \cdots sk$ of steps on $\mathcal{G}$ such that, for $i = 1 \cdots k-1$, $class_2(s_i) = class_1(s_{i+1})$.

Given two class nodes n_c (*picked class node*) and $\hat{n}_c$, a path p starting in n_c and ending in $\hat{n}_c$ defines a GA of n_c as a function mapping each instance x of n_c onto a set of instances of $\hat{n}_c$.

DEFINITION 3.2: [Generalised Attribute] Let $\mathcal{G}$ be a typed-graph, n_c a class node of $\mathcal{G}$ and p a path on $\mathcal{G}$ such that $first(p) = n_c$; the GA of the class node n_c associated to p is a function γ_p: $m(first(p)) \rightarrow \wp(m(last(p)))$ mapping every object $x_0 \in m(n_c)$ to a subset of objects of the last class node of p, $m(last(p))$.

A GA can be either ***single valued*** or ***multivalued*** depending on the cardinality constraints of the role nodes involved in the path. Since a *GA* is a function γ_p returning a set of objects belonging to $m(last(p))$ we will say that γ_p has a ***type*** that is the type of $last(p)$.

GAs encapsulate both implicit and explicit relationships among objects within each object. Thus, by means of the GAs, each object provides a view of the the entire underlying database from the perspective of the object. For example, a GA of the class node person can be "All the hospitals located in the same city where the person lives" (GA3 in Fig. 2). 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 obtained more easily by observing the GAs of hospital and city. Figure 2 also shows the definition of two other generalised attributes, GA1 and GA2, of person.

3.1 Query Specification

The query language of QBI is based on the select-project paradigm: a query is expressed by first defining the conditions that determine a subset of the chosen class node (selection) and then specifying those printable GAs that are going to be part of the output result (projection). A visual query specification is translated internally into a triple $< n_c, Cond, Show >$, where n_c is a class node, *Cond* is a selection condition and *Show* is a set of GAs of n_c to be printed out. Such a query will result in a subset of n_c, say $\hat{n}_c$, containing all the objects of n_c satisfying *Cond*. That is, $\hat{n}_c$ represents a ***derived class*** from n_c and as such it is added to the typed-graph by connecting it to n_c with an ISA edge. Derived class nodes as well as ***primitive class*** nodes that correspond to classes actually stored in the database, can be used subsequently in the composition of more complex queries.

The selection condition is expressed using the GAs of n_c (picked class node) or other class nodes. Each ***atom*** (i.e., logical clause) involves operands of the same type (***type compatibility constraint***) connected by an appropriate operator. Operators can be chosen from among a set of operators associated with the operands type.

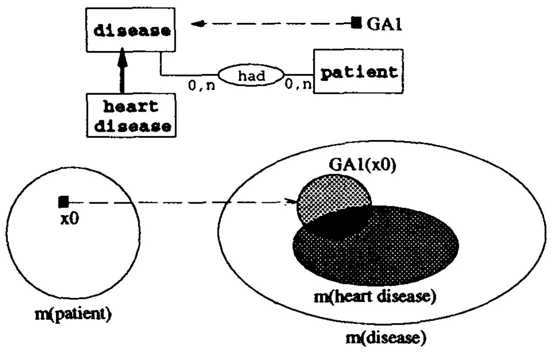


Figure 3: A selection condition

Consider, for example, a medical database dealing with patients and diseases. Suppose there are the class nodes Patient, Disease and a subclass of this node Heart disease (Fig. 3). A multivalued GA of the class node Patient will be "All the diseases the patient has had" (GA1). Suppose the user is interested in all the patients that have had heart diseases; the selection part of this query will be composed of 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.

3.2 Finite Object Schemas

A GA has been defined above as a path in a typed-graph. A path can be cyclic representing a useful property, e.g., `People living in the same city where the person lives`. As a consequence the set of all possible GAs, $\Gamma(n_c)$, associated to a class node n_c is unlimited.

Since not all paths are equally meaningful, and in order to cope with infinitely long (cyclic) paths, QBI defines a *semantic distance function* on paths which returns a value for each path representing the meaningfulness of the corresponding GA. A finite set of GAs of an object is constructed by considering only those GAs that are "meaningful enough" for the specification of a query. The semantic distance is expressed in terms of various aspects of the structure of the GA such as the length of the path, the number of cycles, inclusion/exclusion of specific paths, cardinality constraints as well as statistical information on the underlying database.

DEFINITION 3.3: [Semantic Distance Function] Let γ_P be a *GA* of a class node n_c. The *Semantic Distance Function Semd*: $\Gamma(n_c) \rightarrow \Re$ maps γ_P to a real value $Semd(\gamma_P)$ that represents how much γ_P is semantically distant from n_c.

Given a (monotonically increasing) semantic distance function *Semd* and a threshold value s, the finite object schema (finite *GA* set) of n_c, is defined as:

$$\overline{\Gamma}_{Semd}(n_c) = \{\gamma_P \in \Gamma(n_c) \mid Semd(\gamma_P) < s\}$$

The value for the threshold can be determined by considering the constraints on the system, like response time and available memory. In our QBI prototype (Section 4), we found a threshold that permitted the calculation of about ten thousand of GAs per class.

4 The Prototype

A prototype of QBI has been developed in C for the MS-Windows environment using the toolkit XVT, at the University of Rome "La Sapienza" and it is tailored to provide access to a radiological database [9]. The radiological database contains information about doctors, patients, hospitals, lab tests, radiological images etc. The sise of the QBI prototype itself is only 0.5 MBytes whereas it stores less than 100 KBytes of intensional information about the radiological database. Currently, the GA evaluator of the QBI prototype is being enhanced at the University of Pittsburgh.

4.1 Visualization

In the prototype, visualization of both (primitive and derived) classes and GAs is based on icons and automatically generated natural language text.

It is worth noting that the ability of expressing the whole information content of a database schema in a set of classes and attributes, allows the adoption of a pure iconic paradigm for visualizing the database structure. As a matter of fact, the major limitation of iconic query systems [3], that is the visual representation of relations among concepts, in QBI is not present. The only visual relation that needs to be shown is the containment relation between the icon representing a class and the icon set of its GAs.

An icon consists of several *graphical items*:

- *Picture*. The picture is an image conveying a metaphorical meaning for the class to be represented. For GAs, the picture is the same of the last node of the path.
- *Description*. The description is an automatically generated natural language sentence describing the GA or the class [4]. It plays an essential role for disambiguating the meaning of the icons.
- *Label*. Since description is not always visualized, a label is added to each icon to distinguish on the video display the objects having the same picture.
- *Shape*. The shape is a geometric figure having an outline that allows only the combination of similar shapes. The shape is useful for representing compatibility constraints among types of objects.
- *Frame*. The frame is a stack of shapes indicating the cardinality of the GA or of the class represented.

4.2 Interface Description

Three windows composing the QBI interface are referred to as the *Workspace Window*, the *Query Window* and the *Browser Window* (Fig. 4).

Workspace Window

This window contains of both the primitive and derived 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 at an icon corresponds to selecting the node n_c of a query.

Query Window

The query window is activated when a class icon in the Workspace is selected. In the query window the user composes the selection condition *Cond* and the projection *Show* by dragging and arranging icons in the appropriate window spaces:

- **Conditions Space:** In this space the user builds both the atoms of a selection condition and the condition itself. Atoms can be combined together, according to a positional convention, to form the boolean expression representing the selection condition.
- **Show Space:** The Show Space contains the GAs the user chooses to view in the output result.
- **Description Space:** This space contains a natural language description of the class being defined. The description is automatically generated and dynamically updated whenever the selection conditions change.

Also, within the query window, the GAs that are semantically very close to a picked class are displayed (by default) in the show space. This (*initial GA set*) includes all GAs having a semantic distance less or equal to a constant threshold called *ER threshold*. The ER threshold identifies an initial GA set constituted by the same attributes that would appear in an equivalent Entity Relationship representation of the database.

Browser Window

The prototype provides a facility for querying an object schema, i.e., browsing the GA set of a class. By means of the browser, a user can locate the GAs of a picked class not contained in the initial GA set, that are necessary for composing a query. The set of GAs is sorted by their semantic distance so that the most meaningful GAs are shown first. Each GA is represented by an icon and a natural language sentence describing it. A user can drag these icons from the browser window into either the Condition or the Show space of the Query window.

The GA set can be composed of thousands of elements, depending on the semantic distance threshold used for determining the finite GA set. Thus, the manual browsing of the GA set can be a non-trivial task if the user is interested in semantically distant GAs. For this reason, the browser provides a set of *metaquery operators* that permit the specification of filter conditions on the GA set. In this way, it allows the user to restrict the search of the desired GAs within a smaller set. In particular it is possible to express the following metaquery conditions that are combined in a conjunctive expression.

- **Single:** *Select single valued GAs only.*

- **Printable:** *Select printable GAs only.*

- **Key:** *Select only those GAs that represent a key for the picked class.*

  For example, it may be the case that the GAs "`Name of the city`" or "`Patients living in the city`" identify an instance of the class `City`; in this case, if the user chose the key metaquery condition, they would be selected together with the other keys of `City`.

- **Type:** *Select GAs of a certain type only.*

  That is, this metaquery condition selects all the paths between the picked class node and the node corresponding to the selected type. It is activated by dragging an icon from the Workspace into the Type space in the Browser window.

- **Talk about:** *Select GAs "talking about" a certain set of types.*

  The metaquery selects only those paths that pass through the nodes associated with the specified types. The visual interaction is similar to the previous one: the user drags into the Talk About space the icons corresponding to types the GAs have to include.

- **Don't Talk about:** *Select GAs "that do not talk about" a certain set of types.*

  The mechanism is similar to the previous one.

5 Querying a Radiological Database

In this section, we will describe how QBI can be used to query a Radiological Database. We will use screen dumps taken from the prototype to illustrate the various steps of a query.

5.1 Metaquerying

First, a user selects the database schema to be considered for querying, in our example a radiological database. In response, the workspace window appears on the screen containing a set of icons corresponding to primitive classes of objects (Fig. 4).

Suppose the user is interested in obtaining more information on the class `person`. By pointing the corresponding icon, the query window appears. In order to explore the relationships among the class `person` and the other classes the user activates the browser window. By observing the top of the list of GAs the user can have an immediate perception of the most meaningful attributes of `person`.

In the current version of the prototype, the threshold value for determining the finite set of GAs gives rise to several thousands of GAs for each class node. As a consequence, a user interested in very distant properties of the class `person` can easily explore these properties with the help of the various selection operators of the browser. For example, by dragging the icon `city` in the **Type** space of the browser window (Fig. 4), the user selects only those GAs of `person` having the type `city`. If the user is interested in all the GAs that *talk about* `city`, the same icon will be moved in the **Talk about** space (Fig. 5).

In order to better understand and explore the information contained in the database, the operations we have described thus far can be applied to other classes.

5.2 An example of a query

In this section, let us assume that a user is interested in determining the set of doctors living in the same city in which they work. The result of this query is a subset of the class `doctor` that can be saved as a derived class. In order to build the derived class it is necessary to specify the selection condition *Cond*: *The city where the doctor works is equal to the city where the doctor lives. Cond* can be specified by connecting the two GAs: `City where doctor x lives` and `City where doctor x works` with the connective `Is equal to`.

The first GA is immediately found by scrolling the list in the Browser window. This GA is dragged by the user from the Browser window into the condition space of the Query window. As far as the second GA is concerned, the user needs to perform a metaquery on the GA set of `doctor` by dragging the icon `city` in the **Type** space of the Browser window (Fig. 6). The first GA shown in the list represents the "best" connection between doctor and city and it coincides with the GA the user was looking for, that is: *City where a hospital is located. Such a hospital is the hospital where doctor x works.*

The second GA is then dragged into the condition space and the two GAs are "attached" together; since they have the same type (that of city), their shapes allow this operation to be performed. Once the two GAs have been attached together, a dialogue box containing a set of valid connectives appears. By choosing the equality connective the user ends the selection part of her/his query. In the description space a sentence explaining the selection query is automatically added (Fig. 7).

5.3 A second query

Suppose the user is interested in determining all the patients that have been examined by all and only the Lucky Doctors. For each patient the user wants to determine name, sex, birthdate and the set of radiological images contained in all the examinations made by the patient.

The selection query is performed by first picking the GA: Doctors that have assisted the patient and then by combining this GA with the derived class Lucky Doctors that was defined in the previous query session. Since both the operands of the selection predicate represent a set of objects, a set oriented connective must be used, in particular the set equality operator. Note how the possibility of manipulating multivalued attributes and the possibility of specifying set oriented conditions, allows a very easy way of expressing conditions that involve universal quantification.

For the projection query, the user does not have to pick the name, sex and birthdate because they must be already in the Show space; as far as the radiological images are concerned, the user simply drags the attribute from the browser into the Show space.

6 Conclusions and Future Work

In this paper, we have presented QBI, a visual query language and system based on the Complete Object Model. The goal of the system is to provide a language that is general and easy-to-use by both unsophisticated and expert users. It defines two simple, yet rich primitives, namely the notion of object classes and generalized attributes that are amenable to selection and projection.

We are equally interested in extending both the theoretical and practical aspects of this work. In the area of theory, we are currently working on formally showing that our query language based on the notion of Generalized Attribute is equivalent to domain relational calculus. We are also working on the development of formal framework which can be used to characterize the tradeoffs between expressive power and usability of a query language. We plan to use this framework to compare QBI with other visual query language systems.

Acknowledgements

We would like to thank S.K. Chang for inspiring this work, and S. Pavani and L. Saladini for their suggestion on early versions of this paper. This work was conducted when the second author was a Visiting Professor at the University of Rome "La Sapienza".

References

[1] Abiteboul S. and A. Bonner. Objects and Views. *Proceedings of the International Conference ACM-SIGMOD*, Denver, Colorado USA, 238-247, 1991.

[2] Angelaccio M., T. Catarci and G. Santucci. QBD*: A Graphical Query Language with Recursion. *IEEE Transactions on Software Engineering*, 16(10):1150-1163, 1990.

[3] Batini C., T. Catarci, M.F. Costabile and S. Levialdi. Visual Query Systems. *Technical Report No.04.91. Dipartimento di Informatica e Sistemistica, Universita' di Roma "La Sapienza"*, 1991.

[4] Bono G., P. Ficorilli. Natural Language Restatement of Queries Expressed in a Graphical Language. *Proceedings of the 11th International Conference on Entity-Relationship Approach*, Germany, 1992.

[5] Catarci T. and G. Santucci. Fundamental Graphical Primitives for Visual Query Languages. *Information Systems*, 3(18), 1993.

[6] Chen P.P. The Entity Relationship Model toward a Unified View of Data. *ACM Transactions on Database Systems*, 1(1), 1976.

[7] Groette I.P. and E.G.Nillson. SICON: an Icon Presentation Module for an E-R Database. *Proceedings of the 7th International Conference on Entity Relationship Approach*, Roma, Italy, pp. 271-289, 1988.

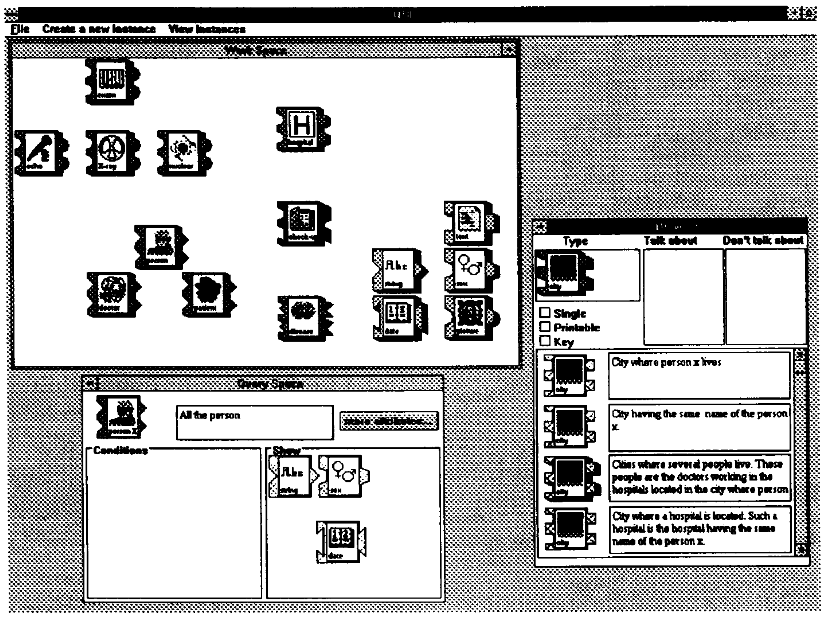
[8] Maier D., J.D. Ullman. Maximal Objects and the Semantics of Universal Relation Databases. *ACM Transactions on Database Systems*, 1(8):1-14, 1983.

[9] Massari A., S. Pavani, L. Saladini. QBI:An Iconic Query System for Inexpert Users. *Proceedings of the Workshop on Advanced Visual Interfaces*, Bari, Italy, pp. 240-242, 1994.

[10] Tonomura Y., S.Abe. Content Oriented Visual Interfaces Using Video Icons for Visual Database Systems. *Proceedings of the IEEE Workshop on Visual Languages*, Roma, Italy, pp. 68-73, 1989.

[11] Tsuda K., M.Hirakawa, M.Tanaka, T.Ichikawa. Iconic Browser: An Iconic Retrival System for Object-Oriented databases *Journal of Visual languages and Computing*, 1(1):59-76, 1990.

[12] Zdonik S.B., D. Maier, eds. Readings in Object-Oriented Database Systems. *Morgan Kaufmann*, 1990.



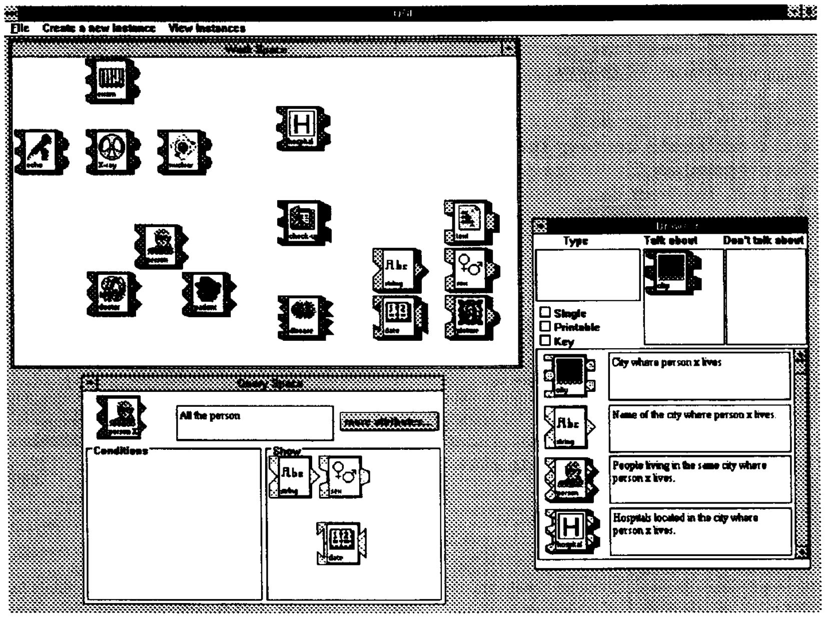
City where person x lives

City having the same name of the person x.

Cities where several people live. These people are the doctors working in the hospitals located in the city where person

City where a hospital is located. Such a hospital is the hospital having the same name of the person x.

Figure 4: Metaquerying



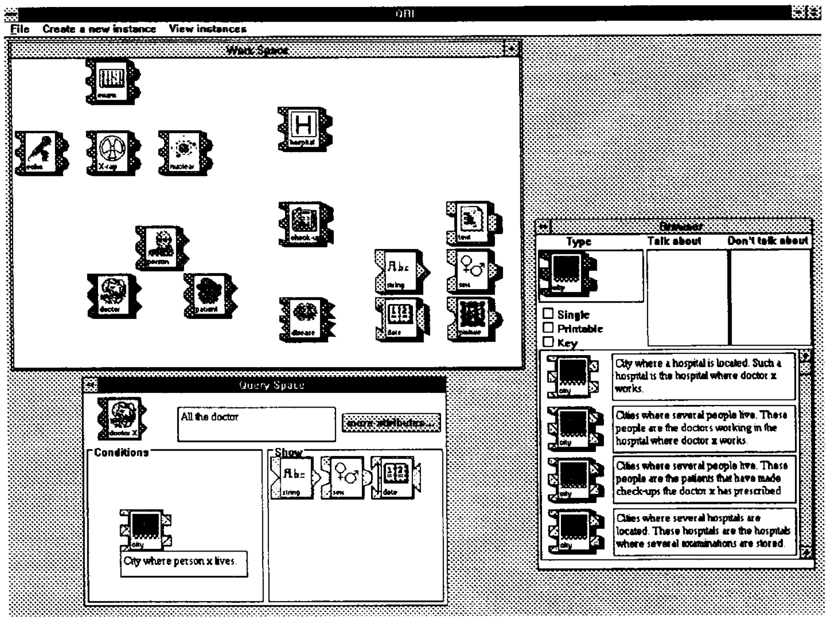
City where person x lives

Name of the city where person x lives.

People living in the same city where person x lives.

Hospitals located in the city where person x lives.

Figure 5: Metaquerying (cont.)



All the doctor

City where person x lives.

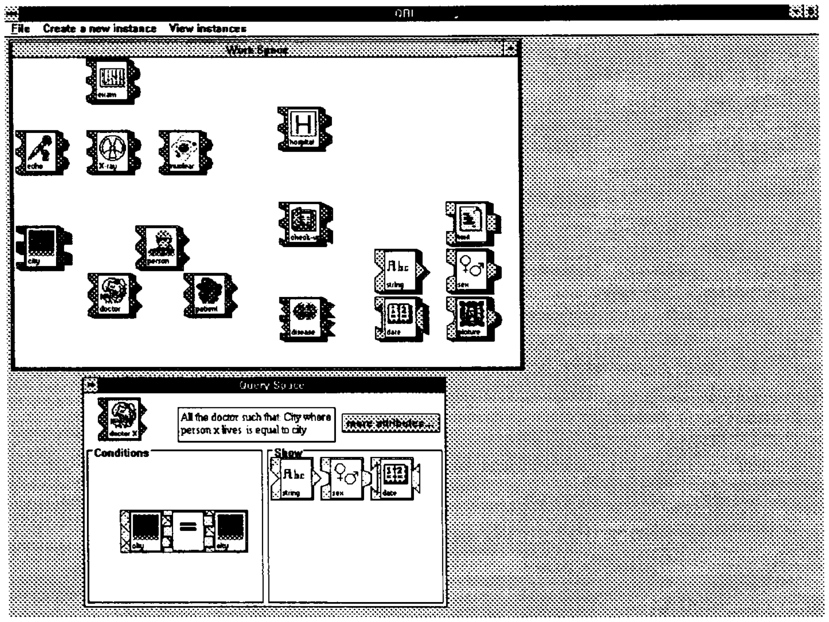
City where a hospital is located. Such a hospital is the hospital where doctor x works.

Cities where several people live. These people are the doctors working in the hospital where doctor x works.

Cities where several people live. These people are the patients that have made check-ups the doctor x has prescribed

Cities where several hospitals are located. These hospitals are the hospitals where several examinations are stored.

Figure 6: A first query



All the doctor such that City where person x lives is equal to city

Figure 7: The selection condition