Comparative Ease of Use of a Diagrammatic Vs. an Iconic Query Language

*Albert N. Badre**, *Tiziana Catarci*°, *Antonio Massari*° *and Giuseppe Santucci*°

*The College of Computing
Georgia Institute of Technology
Atlanta, Georgia - USA
badre@cc.gatech.edu

°Dipartimento di Informatica e Sistemistica
Università degli Studi di Roma "La Sapienza"
Via Salaria, 113 - 00198 Roma, Italy
[catarci/massari/santucci]@infokit.dis.uniroma1.it

Please address all the correspondence to:

Tiziana Catarci
Dipartimento di Informatica e Sistemistica
Universita' di Roma "La Sapienza"
Via Salaria, 113
00198 Roma - Italy
tel.: +39-6-49918331
fax : +39-6-49918331 or +39-6-85300849
e-mail: catarci@infokit.dis.uniroma1.it, tic@cs.brown.edu

Work partially supported by the EC under the Esprit Project 6398 "VENUS".

Abstract

The importance of designing query systems which are effective and easy to use has been widely recognized in the database area. Also, it is well known that the adequacy of a system should be tested against actual users in a well monitored experiment. However, very few such experiments have been conducted. The objective of our study is to measure and understand the comparative ease with which subjects can construct queries in two kinds of visual languages, one diagrammatic and the other iconic. More specifically, we are interested in determining if there is significant interaction between: 1) the query class and the query language type; and 2) the type of query language and the experience of the user.

Experimental results indicate that the effectiveness of a diagrammatic or an iconic query language varies depending on the classes of queries and the kinds of users. This supports the opinion that an interface offering to the user various visual representations and query modalities is the most appropriate for a wide set of users and applications.

Keywords: Human-Computer Interaction; Query Languages; Visual Interfaces; Usability; Experiment.

1. Introduction

The problem of easily extracting information from databases has been discussed extensively in the database community. Several query modalities representing alternatives to traditional query languages such as SQL, have been proposed. Most of them are based on the use of visual representations and direct manipulation interaction mechanisms (see [1] for a survey of visual query languages, VQLs, and [2] for the definition of direct manipulation). All of the proposed approaches emphasize in principle the role of the end user in the design life cycle of a query system. In addition, the importance of designing and testing interfaces for ease and effectiveness of use has become widely recognized ([3, 4, 5, 6]).

However, despite this growing recognition, very few empirical studies aiming at testing and validating the effectiveness of various query styles and interfaces have been conducted in the database field. This is also due to the fact that measuring the ease of using a query language is itself a difficult activity. It requires to identify and control a large number of extraneous variables including, among others, the cognitive capabilities and limitations of the user. To capture at least some aspects of ease of use, experimenters have developed a number of different tasks [7]. The most common are query writing and query reading tasks, which are performed by investigating the relationships between database queries expressed in natural language and the same queries expressed in the query system under study. In query writing, the question is: "Given a query in natural language how easily can a user express it through the query language statements? The question for query reading is: "Given a query expressed through the query language statements can the user express the query easily in natural language?". More precisely, what is measured is the *user's performance* when accomplishing such tasks. Such performance is usually expressed in terms of 1) *accuracy*

of query completion (i.e., user's correctness rate when writing the queries) and 2) *user's efficiency* (i.e., time spent to complete a query).

Two different classes of experiments have been reported in the literature: one aiming at evaluating the ease of use of a single language, the other aiming at establishing which is the easiest to use among two or more languages. Reisner [7] reviews most of the experimental research on traditional query languages covering mainly work done on SQL and QBE. Recently, QBE and SQL were again compared, taking into account several factors, such as the use of the same database management system, a similar environment, etc. [8]. It is interesting to note that in this new experiment, the query language type affected user's performance only in so-called "paper and pencil" tests, in which case QBE users have higher scores than SQL users. In on-line tests, the user's accuracy was not affected by the type of the language adopted, but the user's satisfaction was much greater with QBE, and his/her efficiency much better. Other earlier surveys on experiments in the database field include papers by Shneiderman [9, 10] and Thomas [11]. A more recent work is presented in [12]. This study concentrated on a language based on the querying technique called "dynamic query". Given a query, a new query is easily formulated by moving the position of a slider with a mouse, and the display of the resulting data is dynamically modified. The language was tested against two other query languages, both providing form fill-in as the input method, but having different output methods. The hypothesis that the dynamic query language would perform better than both other ones was confirmed by the experiment results.

As we said above, a common goal of such empirical studies is to provide a quantitative basis for the comparative effectiveness and ease of use of a given query language and interface. Frequently, they aim at verifying some hypothesis, which could come from either the intuition of the authors or, more often, the simple observation of people using the system[1]. The general approach used in most such studies includes: 1) defining precisely what one is to measure; 2) developing a task for users to perform; and 3) measuring relevant parameters of user's performance (i.e. error rate, time to complete a task, etc.). A few tens of subjects are usually involved in the experiments (ranging from 18 in [12] and 20 in [14] to 65 in [8]).

In a recent empirical study, Catarci and Santucci, compared the visual query language QBD* (Query by Diagram) with SQL [15]. The results of the study confirmed the hypothesis that a visual language is easier to understand and use than a traditional textual language. In this paper, using primarily the same type of query writing task used earlier by Catarci and Santucci, we describe the outcome of an experiment to compare the effects on user's performance of two different query languages, one iconic and the other one diagrammatic. The independent variables of the experiment are user's skill and query classes. Since we conjectured that there is a significant relationship between the independent variables and the visual query language used, we designed our experiment to test such conjecture.

1 Talking to real users, visiting their working environment, observing how they approach their task are activities which form the basis of usability engineering [3].

The two visual query languages, developed at the Dipartimento di Informatica e Sistemistica, are QBD* [16, 17, 18] and QBI (Query by Icon) [19, 20]. As we will clarify below, the two languages differ in both the adopted visual formalism (diagrams vs. icons) and the main query strategy (navigation vs. composition). The visual formalism and the query strategy are two fundamental features of VQLs, since they both determine the nature of the human-computer dialogue, whose improvement is the primary goal of VQLs [1, 6, 21]. Being both diagrams and icons, as well as navigational and compositional query strategies used in most of existing VQLs (see [1]), the two systems under analysis can be also seen as representative of larger VQL classes. To the best of our knowledge, this is the first experiment comparing the ease of use of two VQLs based on different visual formalisms and query strategies.

The actual experiment was preceded by a teaching period. We tried to make as many of the conditions of the experimentation of the two languages the same as we reasonably could (i.e., teaching method, exam questions, exam procedure, method of scoring). We also examined some background information about the various users in order to divide them into equivalence classes and to assign to each language the same set of classes. This is commonly done in experiments in order to minimize the probable effects of uncontrolled extraneous variables leading to unreliable results

The paper is organized as follows. In Section 2, we summarize the elements of QBD* and QBI. In Section 3, we identify and classify the two independent variables, namely user's skill and query classes. In Section 4, we describe the design of the experiment, the results obtained and our interpretation of the findings. Finally, conclusions are drawn in Section 5.

2. The Prototype Visual Query Systems

In the following we summarize the basic features of the two visual query systems we have tested, namely QBD* and QBI. From an internal point of view, both systems are based on a semantic object-based data model. In particular, QBD* is based on the Entity-Relationship (ER) model [22], while QBI uses the Binary Graph Model [20, 23], which corresponds to the Graph Model introduced in [24] restricted to consider binary relationships only. It has been shown in [25] that a trivial mapping exists between the ER model and the Graph Model, so that we can consider the two systems actually based on the same internal data model, namely the Graph Model.

The Graph Model allows one to define a database D in terms of a triple $<g, c, m>$, where g is a Typed Graph, c is a set of suitable Constraints, and m is the corresponding Interpretation. The schema of a database is represented in the Graph Model by the Typed Graph and the set of Constraints. The instances of a database are represented by the notion of Interpretation. The set c may be specified by using a logic-based constraint language, which is intended to be exploited by the system designer in order to specify constraints on and meaningful properties of the nodes represented in the Typed Graph. A node in a Typed Graph can be either a *class-node* representing a class of objects or a *role-node* representing a relationship between two classes, whereas an edge connects a class-node to a role-node. Moreover, class-nodes are partitioned into two

subsets: the set of *printable class-nodes* (concrete classes) and the set of *unprintable class-nodes (abstract classes)*. The former represent set of objects that are actually values of distinguished domains (real, integer, char, etc.), whereas the latter represent sets of objects denoted simply by object identifiers. We call *path* in a Typed Graph a sequence of adjacent class-nodes and role-nodes always starting and ending with a class-node. Given a path it is possible to associate with it a multivalued function which maps every object belonging to the Interpretation of the first class-node to a set of objects of the last class-node of the path. Note that the definition of path does not exclude the presence of cycles.

From the user's point of view, the two systems mainly differ in the visual formalism they adopt for representing the database schemata, and in the basic query strategy. QBD* is based on the use of diagrams, namely ER diagrams, while QBI adopts icons. Both formalisms present complementary advantages and disadvantages. For instance, while diagrams are particularly effective in representing the relationships existing between concepts, which are directly mapped into correspondences among the diagram elements, icons are powerful in resembling objects of the real world having an intrinsic visual representation [26].

QBD* allows the user to query the database by schema navigation**,** i.e. by concentrating on a concept and moving from it following the diagram links in order to reach other concepts of interest. QBI works by composition and grouping of icons, i.e. by interpreting simple actions, such as the selection or the dragging of an icon, as a particular chain of operations on the database.

We will see in Section 4.4 how such differences are reflected in the subjects' behavior when composing a query.

*2.1 QBD**

QBD* is a visual query system based on diagrammatic representations of ER schemata describing the underlying (relational) databases. It balances a high expressive power with a noticeable facility of use. Concerning the expressive power, it has been proved to be relationally complete. Moreover, it includes a significant class of recursive queries (transitive closure) and handles an extension of the relational algebra set-oriented operators that we call *generalized set-oriented operators* [27]. The ease of use has been reached through a fully graphical environment. In that environment the user interacts with the system mainly with a mouse-like device, using the keyboard only when necessary.

The general structure of the QBD* query is based on the location of a distinguishing concept (entity or relationship), called *the main concept*, which can be seen as the entry point of one or more subqueries. These subqueries express possible navigation from the main concept to other concepts in the schema. The attributes belonging to each subquery are determined by the following strategy: the attributes of the main concept are automatically considered (unless explicitly deleted by the user), while the other ones are shown in the result only if requested by the user. The presence of a main concept associates a type with each subquery. As an example, if the main concept is the entity `person`, the result of the query will be a set of people (possibly enriched with attributes coming from other concepts), no matter what kind of subsequent

operations the user performs to specify it. The subqueries can be combined together by applying several set-oriented operators to two or more subqueries derived from the same entity.

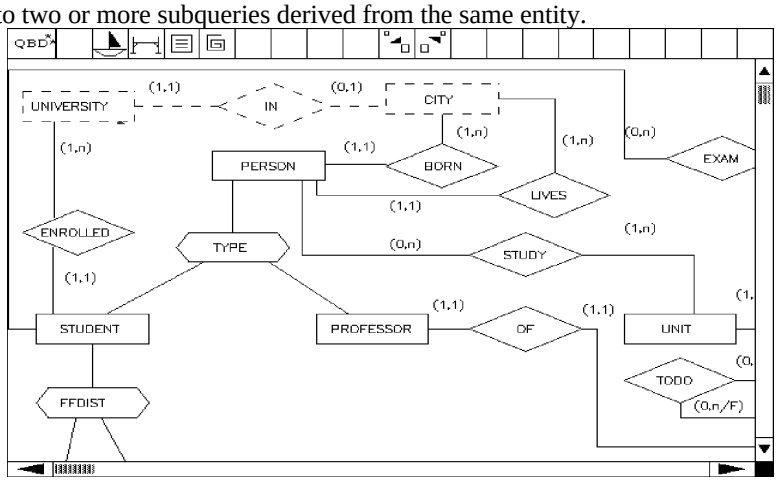


Figure 2.1: Schema navigation through existing paths

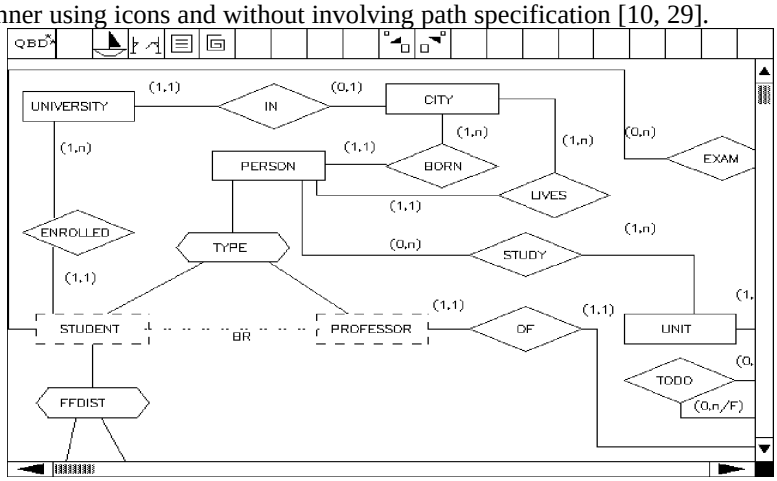
Once the main concept has been selected, two types of strategies are available for navigating in the schema. The first one allows the user to follow existing paths, so exploiting predefined relationships among data. In the example in Figure 2.1, through the path (`University, In, City`) (plus the condition `city.name="Rome"`, in a separate window, not shown in the figure) the user asks for all the Universities in Rome. Figure 2.2, instead, refers to the second strategy and corresponds to the first part of the answer to query Q4 (see Section 3.2), in which the user searches for all professors whose last name is equal to the last name of a student enrolled for a Rome's University. To find out such professors, the user has to build a new relationship between the entities `professor` and `student` (see Figure 2.2.a). After that, the user has to specify the comparison condition used to build the new relationship, that is `professor.last_name=student.last_name`. Comparison conditions, as well as generic conditions on the attributes, are expressed by means of a simple window mechanism including a suitable set of icons (see Figure 2.2.b). Roughly speaking, we can say that a path on the schema corresponds to an ordered sequence of natural joins[2] among the pairs <entity, relationship> constituting the path, followed by final selection and projection. The explicit presence of relationships releases the user from the need of looking for concepts like "foreign keys" and the system can perform in the right way the join operations. On the other hand, comparing two concepts via a new relationship corresponds to a theta-join between the two involved entities, and it is up to the user the specification of the theta-condition.

2.2 QBI

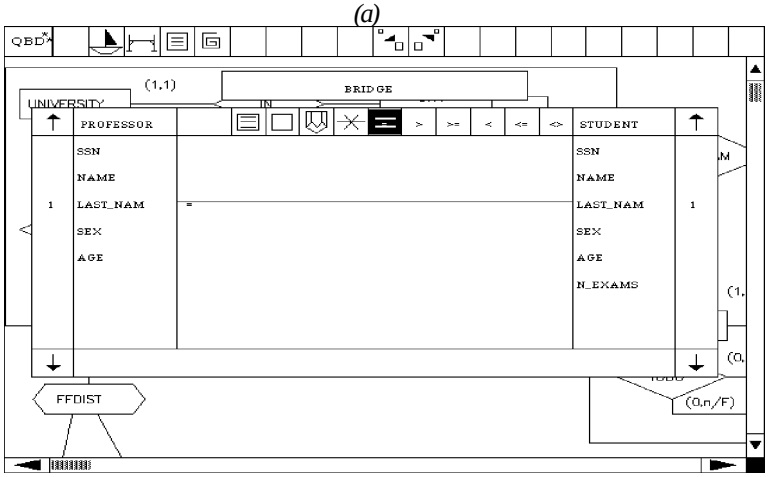
QBI is a system that allows users to query and understand the content of a database by manipulating icons. QBI was originally conceived as an interface for a distributed database of radiological images [28] and subsequently developed into a general purpose query processing facility that is domain independent. It

[2] We mean here the natural join and theta-join operators as defined in relational algebra [38].

provides *intensional browsing* through *metaquery* tools that assist in the formulation of complete queries in an incremental manner using icons and without involving path specification [10, 29].



(a)



(b)

Figure 2.2: Schema navigation through the building of a new relationship (2.2a) on the basis of attribute comparisons (2.2b)

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. Users are presented with database abstractions called *complete objects*, i.e. completely encapsulated objects, similar to the *universal relation* abstraction in relational databases [30]. A complete object has several properties called *Generalized Attributes* (GAs in the following). For example, the set of `exams` made by a `student` is treated as a GA of `student` in a way similar to that of simple attributes such as its `name` or `birthday`. In QBI, each class provides a view of the whole underlying database from its own viewpoint. 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 (selection) and then specifying those GAs that are going to be part of the output result (projection). The result of a query can be used subsequently in the composition of more complex queries.

Three windows forming the QBI interface are referred to as the *Workspace Window*, the *Query Window* and the *Browser Window*. In figure 2.3 the three windows of QBI containing the icons related with the experiment database (i.e., students, professors) are displayed.

Workspace Window. This window displays 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 an icon corresponds to selecting the viewpoint class-node of a query. In figure 2.3 the selected icon is `professor`.

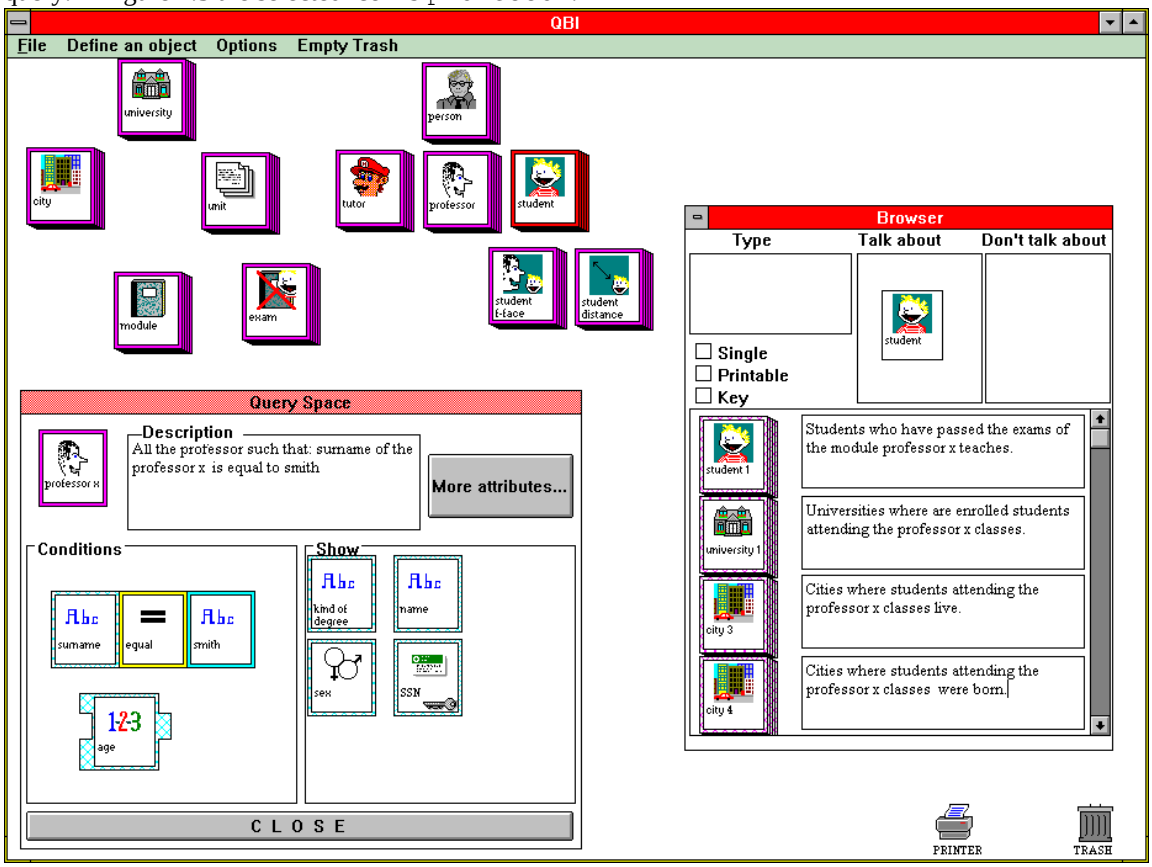


Figure 2.3: The three QBI windows

Query Window. In the query window the user composes both the selection, *Conditions* subwindow, and the projection conditions, *Show* subwindow, by dragging and arranging icons. Referring to figure 2.3 the condition is: `"surname of the professor" equal to "Smith"`, which is composed by the GA `"surname of the professor"` (the `surname` labeled icon), the constant value `Smith` and the connective `equal`. Note that the GA `"age of the professor"` (the `age` labeled icon) has been dragged in the condition space; this GA could be combined with either another GA or a constant value to create a new atomic condition.

Browser Window. The browser window contains the GAs of the chosen class-node (`professor` in Figure 2.3). This set can be composed of thousands of elements. The metaquery operators allow the user to search the desired GAs within a smaller set. Among the various metaquery operators, it is possible to select all the GAs having a certain type. Moreover, it is possible to select either a single valued GA only, or

those GAs that include in their paths a certain class-node, i.e. whose description "talk about" a certain concept. For example, the browser window shown in figure 2.3 contains the icon `student` in the Talk About space, whereas `professor` is the selected icon. As a consequence, only the GAs of `professor` talking about `student` are presented. Note that on the left of each GA icon, a natural language sentence is displayed. This sentence is automatically generated by QBI in order to disambiguate the meaning of the GA.

3. The Experiment Factors

In this study, we conjecture that there is a significant relationship between the independent variables, user skill and query type, and the visual query language used on the time and accuracy of query construction. Therefore in the experiment, users of different skill levels will be asked to construct different types of queries. This choice reflects (and aims to validate) the current literature trend which relates the appropriateness of the visual representations to the level of the user's skill and to the kind of queries s/he wants to perform (see, e.g., [31, 32, 33]).

3.1 The User Classes

A few user classifications have been proposed in the database literature (e.g., [31, 33, 34]). Each of them identifies a certain number of features which permits the labeling of a homogeneous group of users. The number and the kinds of groups differ depending on the specific classification. However, there is at least a general agreement on the initial splitting of the users into two large groups: those who have had a certain instruction period and have database knowledge, and those who do not have specific training in databases. In our experiment we call those two groups skilled and unskilled users respectively. Our unskilled user is roughly characterized by the following features: 1) s/he interacts with the computer only occasionally, 2) has little, if any, training on computer usage, 3) has low tolerance for a formal query language, 4) is unfamiliar with the details of the internal organization of an information system. Usually this user does not want to spend extra time in order to learn how to interact with a query system, and finds it irritating to have to switch media, e.g., to manuals, in order to learn how to interact with the system. Moreover, the unskilled user wants to know where s/he is and what to do at any given moment of the interaction with the system. Notice that the unskilled user is very similar to Cuff's [34] casual users.

On the other hand, skilled users possess knowledge of database management systems, query and manipulation languages, etc., and often like to acquire a deep understanding of the system they are using.

3.2 The Query Classes

For the purpose of our experiment we need to identify meaningful classes of queries. Several query classifications have been proposed in the literature (see, e.g., [35, 36, 37]). However, those classifications are all based on considering the so-called expressive power of the query language, i.e. the ability of the language to extract meaningful information from the database, and/or the complexity of computing the

query answer. While extremely valuable in general, such classifications are inadequate for our purposes since they do not take into account the user's point of view, i.e. the user's effort when formulating a query. For example, all queries containing selection operations only are traditionally considered as belonging to the same class, despite the kind of formula used for expressing the selection condition (e.g., with or without boolean operators). However, observing the different difficulties the user encounters when formulating such queries, one can easily conclude that they should be in different classes. Whereas, we need a query classification which evaluates in a different way queries with different intrinsic complexities in terms of user-oriented features.

By a preliminary study on users interacting with QBD* and QBI, we observed significant differences in the ease with which they did formulate the query. These differences are strongly dependent on the number of database concepts involved in the query and the way in which they are related. Thus, we conjectured that these two features could be good candidates for query classification. We exploit the notion of path as introduced in Section 2, for formalizing the two coordinates in terms of:

- the maximum *semantic distance* of the paths involved in the query;
- the overall number of *cycles* of the paths involved in the query.

The concept of semantic distance is introduced for ordering the importance of user's access paths. The semantic distance, formally introduced in [19], is a mathematical function modeling both the *meaningfulness* and the *expected utility* of a path for the composition of a query. The meaningfulness of a path can be regarded as the difficulty a user encounters in understanding the path meaning. The expected utility measures "how likely" the path will be used in a query.

More precisely, let P be the set of well formed paths on a Typed Graph, a semantic distance function $SD: P->R$ is a function mapping every path $p \in P$ into a real number $SD(p)$. Given a path p it is possible to derive numerical attributes, called *features* of the path, each one representing a characteristic useful for modeling the semantic distance function. In [19] several features have been introduced and motivated. The most significant are: 1) the length of the path and 2) the number of printable class-nodes included in the path (excluding the last node). These two features have been also used in the present work. For the sake of simplicity, the codomain of the semantic distance function has been partitioned into three ranges, namely L, M, H. The L range comprises those queries whose paths have a length less than or equal to two and no printable inclusion. In the M range we put the paths with a length up to 3 and no printable inclusion, i.e. no printable class-nodes included in it. The H range includes path longer than 3, and/or allowing printable inclusion.

As for computing the second classification coordinate, we recall [19] that every path has associated a function. Such a function is determined by 1) declaring for each class-node of the path an existentially quantified variable ranging on the class-node interpretation, and 2) specifying a set of connection conditions between pairs of consecutive variables, each pair being forced to be included in the interpretation of a role-node. Now, we define the number of cycles in a path p as a function $C(p): P->\mathbb{N}$. In particular:

$$C(p) = length(p) - number\ of\ distinct\ class\text{-}nodes\ in\ the\ path + 1 \qquad (1)$$

where *length(p)* is the length of the path *p* in terms of number of role-nodes included in the path. Referring to the path function, note that when the number of cycles is greater than zero, it follows that there exist at least two existentially quantified independent variables ranging on the interpretation of the same class-node. As a consequence, a query whose formulation requires cyclic paths forces the user to deal with different references to the same class-node.

In the following we describe the six queries used in the experiment. The above features, i.e. length of the path, printable inclusions, and cycles, from now on will be always calculated on the basis of the Typed Graph shown in Figure 3.1, which is the experiment view of a more general Typed Graph depicting the schema of a University database.

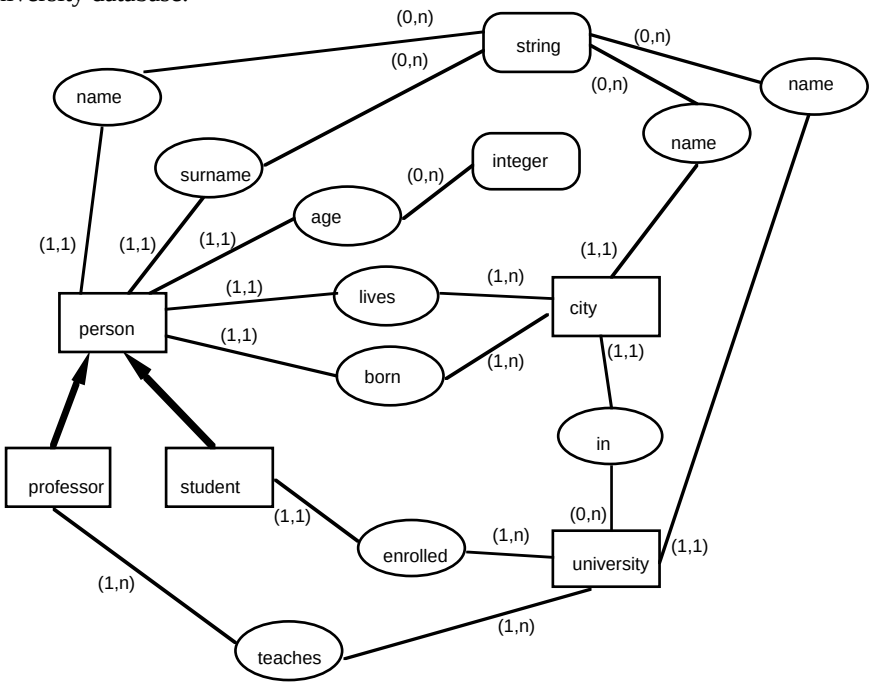


Figure 3.1: Experiment Typed Graph

Query Q1 : "Find out students older than 40". This query involves the path (`Student age integer`) of length 1, which includes one role-node corresponding to the age of the student . The semantic distance of the path is L and there are no cycles.

Query Q2 : "Find out all persons enrolled for and teaching in the same University". The query involves the path `(Person, enrolled, University, teaches, Person)` of length 2. The path has a semantic distance L and there is a cycle (two variables on the class-node `person` are declared within the query expression)[3].

[3] Note that the ISA constraints `Professor` ISA `Person` and `Student` ISA `Person` do not imply the disjunction of the classes `Student` and `Professor` .

Query Q3 : “Find out all students enrolled for a University in Rome”. The query involves a path of length 3 (`Student, enrolled, University, in, City, name, string`). The semantic distance of the path is M and there are no cycles.

Query Q4 : “Find out all professors whose surname is equal to the surname of a student enrolled for a Rome’s University”. The query involves a path of length 4 (`Professor, surname, string, surname, Student, enrolled, University, in, City`) and it does include a printable class-node (`string`). There are no cycles in the query (there are five distinct class-nodes in the path, therefore the expression (1) is 4-5+1). The semantic distance of the path is H.

Query Q5 : “Find out all persons who live in the same city where the University they are enrolled for is”. The query involves the path (`Person, lives, city, in, University, enrolled, Person`) of length 3. The semantic distance is M and there is a cycle.

Query Q6 : “Find out all students with the same surname of professors teaching in the University located in the city where they live”. The query involves two paths: (`Student, surname, string`) of length 1, and (`Student, lives, City, in, University, teaches, Professor, surname, string`) of length 4. While the semantic distance of the first path is L, the semantic distance of the second one is H. As a consequence the overall semantic distance of this query is H and there are no cycles.

		CYCLES	
		0	1
SEMANTIC DISTANCE	LOW	Q1	Q2
	MEDIUM	Q3	Q5
	HIGH	Q6,Q4	

Figure 3.2: The classification of the experiment queries

Figure 3.2 shows the classification of the six experiment queries with respect to the two above coordinates when the codomain of the semantic distance function has been partitioned into three ranges. The same queries have then been grouped into three classes, using the two value partition of the codomain of the semantic distance function. The subjects’ performances as a function of these three classes have been deeply studied in the experiment. In the following we will refer to the three classes as *close-uncyclic* (low-medium semantic distance and no cycles), *close-cyclic* (low-medium semantic distance and cycles), and *far-uncyclic* (high semantic distance and no cycles).

4. The Experiment

The overall objective of our study is to measure and understand the comparative ease with which subjects can construct queries in either the diagrammatic or iconic language. More specifically, we are interested in three questions: 1) given a query in natural language, how easily can a user express it by using either the Diagrammatic or Iconic representation; 2) given different types of queries, is there significant

interaction between the query-type and the visual language used; and 3) given different levels of skill is their significant interaction between skill level and query language used? The experiment is designed to determine if there is significant interaction between: 1) the query class and the query language type as represented in table 4.1; and 2) between the type of query language and the experience of the user as reflected in table 4.2.

4.1 Participants

A total of thirty-two subjects were used in the experiment. The subjects were student volunteers. Sixteen of the subjects were third year Computer Science students. The other sixteen were graduate students in science and engineering. One-half of the graduate students and one-half of the undergraduates were assigned to each of the two query language conditions. In addition, as discussed below, experience with query languages and skill level were also taken into account when assigning subjects to the two language groups (see Table 4.1[4]).

2	**Natural language query -----> Interface query**	
	DIAGRAMMATIC	ICONIC
Skilled users	Group 1S NLQ->DIQ	Group 2S NLQ->IIQ
Unskilled users	Group 1U NLQ->DIQ	Group 2U NLQ->IIQ

Table 4.1: Query language as a function of skill level

4.2 Procedure

A pretest as well as a background questionnaire were administered to all subjects. The purpose of the pretest was to determine each subject's level of knowledge and understanding of the database query languages. The subjects were then assigned to one of two treatment conditions, iconic or diagrammatic. Level of experience with query languages and pretest performance scores were used to equate the two groups. Next all subjects underwent a short training session of equal times in using the diagrammatic or iconic interface environment.

After the training session, each subject was presented with six queries of different levels of complexity in natural language and was asked to construct these queries in one of the two query language environments (see Table 4.2). The order in which subjects were given the queries (close-uncyclic, close-cyclic, far-uncyclic) was counterbalanced across subjects in order to minimize the learning effect.

[4] In Tables 4.1 and 4.2, NLQ, DIQ, and IIQ stand for Natural Language Queries, Diagrammatic Interface Queries, and Iconic Interface Queries respectively.

Natural language query -----> Interface query		
	DIAGRAMMATIC	ICONIC
Close-uncyclic queries (Q1, Q3)	Group 1 NLQ->DIQ	Group 2 NLQ->IIQ
Close-cyclic queries (Q2, Q5)	Group 1 NLQ->DIQ	Group 2 NLQ->IIQ
Far-uncyclic queries (Q4, Q6)	Group 1 NLQ->DIQ	Group 2 NLQ->IIQ

Table 4.2: Interface Type × Query Complexity For Expression as an Interface Query

4.3 Results

Time in seconds to complete a query and accuracy of query completion were used as the two performance measures. In our analysis we were mainly interested in the differences in performance between the subjects using QBD* and those using QBI. We were also interested in determining whether there was a difference in performance as related to the class of queries and the skill level of the user. Table 4.3 shows the overall average accuracy and time scores for QBD* and QBI. The well-known statistical anlysis of the variance (i.e., AN analysis Of VAriance test, ANOVA) yielded no significant difference on accuracy between QBI and QBD* at [F(1,30) = 0.025, P> 0.05. There was however a significant difference on the subject's time performance between the two systems at F(1,30) = 5.73, P < 0.05. At inspecting of the time means, it is clear that the subjects spent on the average less time conducting queries using QBD* than QBI.

		VISUAL LANGUAGE	
		QBD*	QBI
PERFORMANCE	TIME	129	211
MEASURES	ACCURACY	8.7	8.8

Table 4.3: Overall average time and accuracy

Performance as a Function of Skill. Table 4.4 shows time performance for QBI Vs. QBD* as a function of skill level. Again, an ANOVA here yielded a statistically non-significant difference between the two systems for low-skill inexperienced subjects at F(1,14) = 0.343, P> 0.05. On the other hand, there was a significant difference for high skilled people with experience at F(1,14) = 10.30, P <0.05. At an inspection of the means in table 4.4, it is clear that skilled subjects take less time to construct a query using the QBD* system. The reason for this result could very well be that skilled subjects have more experience using

graphs than the unskilled ones. On the other hand the skilled subjects experience in using icons is similar to that of the unskilled group.

		VISUAL LANGUAGE	
		QBD*	QBI
USER SKILL LEVEL	SKILLED	104	218
	UNSKILLED	154	205

Table 4.4 :Average time in seconds to construct a query as a function of skill level

Performance as a Function of Query Classes. Table 4.5 gives a summary of the average time and accuracy performance as related to the different query classes discussed earlier. The accuracy data show no significant difference between the groups as a function of query classes. In comparing group performance for each one of the query classes, ANOVAs yielded significant differences for classes close-cyclic and far-uncyclic at $F(1,30) = 17.77$, $P < 0.05$ for class close-cyclic, and at $F(1,30) = 28.96$, $P < 0.05$ for class far-uncyclic, and no significant difference for class close-uncyclic. At inspecting the means in table 4.5, it is clear that the subjects do better on type close-cyclic query if they use the QBI system. On the other hand, for query type far-uncyclic, the subjects' performance is better if they use the QBD* system.

		VISUAL LANGUAGE			
		QBD*		QBI	
		TIME	ACCURACY	TIME	ACCURACY
QUERY CLASSES	CLOSE-UNCYCLIC	95.5	9.6	145	9.4
	CLOSE-CYCLIC	135	7.5	41	10
	FAR-UNCYCLIC	156	9.1	448	7.2

Table 4.5: Average time in seconds and Accuracy as a function of query classes

4.4 Discussion

Both the accuracy and the response time of the subjects using QBI seem to be highly sensitive to the semantic distance of the query paths. More specifically, the two queries involving a path with a semantic distance value H, Q6 and Q4 (see Figure 3.2) respectively, have the lowest degrees of accuracy and the highest response times. On the other hand, QBD* manifests a substantial independence both on the accuracy and on the response time with respect to the semantic distance of the paths involved in the query expression.

Whenever a query contains a cycle (Q2 and Q5), expressing it using QBD* is less accurate and take a longer time. QBI is not affected by the presence of cycles, on the contrary, it yields the best results both in

terms of accuracy and response time when cycles are present. In the case of query Q5, the subjects' response time for QBD* is four times as high as the time for the QBI subjects. This difference can be motivated by observing that the attributes belonging to the same concept are reached by following different paths.

		VISUAL LANGUAGE			
		QBD*		QBI	
		TIME	ACCURACY	TIME	ACCURACY
QUERIES	Q1	80	9.4	134	9.5
	Q2	84	7.1	27	10
	Q3	111	9.8	156	9.25
	Q4	179	9.3	599	7
	Q5	187	7.9	55	10
	Q6	134	9	298	7.5

Table 4.6: Average response Time and Accuracy per query

The previous results can be justified as follows: the sensitivity of QBI to the semantic distance of the paths involved in the query is a direct consequence of the way in which QBI forces the user to think about the query. More specifically, in QBI the user should think about the query predicate as a selection involving generalized attributes. A generalized attribute is a multivalued function that is associated with a path on the underlying semantic schema (see Section 3.2); therefore, either when the path is not composed of all predefined relationships or when the path is long, the user finds it difficult to understand the corresponding path function.

In QBI the navigation phase is replaced with the activity of searching the GA corresponding to the path to be specified. The reason why QBI suffers when the path to be found is semantically distant, is not in the browsing activity itself, but in the cognitive process that comes before the browsing activity, that is to think about the whole path in terms of a function, often multivalued, of the generic instance of the selected class. Referring to the query Q4, for example, the user should have thought to find the GA of professor: "`All the students that have the same last name of the professor x`". Even though this attribute is not difficult to understand, we have observed that the major part of the subjects just refused the idea that such a kind of GA could have even existed. It is interesting to report the reaction of surprise all the subjects had when, after having spent minutes in trying to understand how to construct the query, they finally realized that such a strange GA was present in the list. In some sense, this reaction demonstrates that the problem with this query was not how to find the GA, but simply to understand what had to be found.

As far as the query Q6 is concerned, the problems we observed in the QBI subjects were similar to the ones described for query Q4. However, in this case, the cognitive process of understanding which GA had to

be found for building the query, had less influence on the overall performance. In particular, we observed that the major component for the whole time spent by the subjects in finding the attribute "`All the last names of the professors teaching in the University located in the city where student X lives`" was due to difficulties in the metaquery techniques.

The reason why QBD* users perform better when expressing queries characterized by a high semantic distance could very well be that paths in QBD* are built manually. As a consequence, the user perceives the whole path not as a unique complex function, but as a sequence of single steps. Moreover, the user has a total control on the path specification, since it is always possible to backtrack and, once the path has been completed, to change the conditions which have been previously specified for each single step.

Considering the queries containing cycles, we can say that whenever a query involves two attributes belonging to the same concept the QBD* users get much more confused than the QBI ones. Our explanation resides mostly in the way in which the two systems present the attributes to the user. In QBD* the attributes of any concept (entity or relationship), are represented as labels grouped into a form associated with the concept (see Figure 2.2b). As a consequence, when the query is cyclic, the user sees more copies of the same form, belonging to the different occurrences of the same concept met along the path. In order to associate each form with the right concept occurrence, the system automatically adds a number to the attribute labels. However, we noticed that, even using numbers, most users are still confused, not clearly understanding the meaning of the replicated labels.

On the contrary, in QBI a path corresponds to a GA and every GA is visually represented as a different icon on the screen. Therefore, when a query expression contains cycles, the user still perceives a clear distinction among different occurrences to the same concept. In the case of query 5, for example, the path (`Person, lives, city, in, University, enrolled, Person`) was split into two distinct paths: (`city where the person lives`) and (`city where the University the person is enrolled for is`) which were represented by two distinct `city` icons, as a result almost all the subjects did not get confused in constructing the appropriate selection predicate by combining such icons.

5. Conclusion and Future Work

In this paper we compared two visual query languages, QBD* and QBI performing a usability experiment on them. The experiment showed significant relationships among the kind of visual language and the user's skill and query classes, pointing out that the two systems exhibit complementary advantages and disadvantages. On the basis of the figures we got we can say that skilled users perform better using the QBD* system, while no significant difference exists concerning the performance of unskilled users. On the other hand, while considering the query classes we discovered noticeable differences between the class of close-cyclic queries (better performed in QBI) and far-uncyclic queries (better performed in QBD*), such differences involve both time and accuracy and are totally independent from the skill of the users. In

summary, the analysis of the experiment results allows us to say that both systems got good results in terms of effectiveness, even if such results vary depending on the classes of queries and the kinds of users. This supports the widely believed opinion that an interface offering to the user various visual representations and query modalities is the most appropriate for a wide set of users and applications (see, e.g., [21, 31, 32, 39]).

We plan to further investigate on the effectiveness of QBD* and QBI by performing other experiments, at both the University of Rome and the Georgia Institute of Technology, involving a larger number of users and other classes of queries. On the other hand, we will also start to test a prototype of a multiparadigmatic query interface, i.e. an interface offering to the user different visual environments and the possibility to switch among them, which is under development at the University of Rome.

Finally, we want to point out that the results of the experiments led us to make changes in both prototypes. Referring to QBD* it came up by the users' comments that the default option we used in combining two or more attribute conditions (logical AND) was misleading and we removed it. Moreover, the users reported the need of a more uniform way of interacting with the system and we rearranged the user interface in order to present several phases of the query formulation in a homogeneous way in spite of their conceptual difference. As for QBI, we have got extremely valuable hints for improving both the metaquery operators and the query specification mechanism. In particular, we very often observed subjects dragging a GA, say a_1, in the Talk About space with the purpose of selecting only other GAs having a path including a_1. Moreover, different subjects once selected an icon and specified an atomic condition, tried to move the GAs of the selected icon directly on the workspace window. This operation was not allowed in the experiment version of QBI. However, it is consistent with the interface philosophy and has a neat semantics: since the variable ranging on the selected icon is in some way bound by the atomic condition, the corresponding GAs can be regarded as queries themselves. Both these mechanisms have now been implemented and we are observing significant improvements in the ease of use of QBI.

References

1. Catarci T., Costabile M.F., Levialdi S., Batini C. Visual Query Systems for Databases: A Survey. Technical Report SI/RR-95/17 of Dipartimento di Scienze Dell'Informazione, University of Rome "La Sapienza", 1995.
2. Shneiderman B. Direct Manipulation: A Step beyond Programming Languages. IEEE Computer 1983; 16, pp 57-69.
3. Nielsen J. Usability Engineering. Academic Press, San Diego, CA, 1993.
4. Bevan N., Macleod M. Usability Assessment and Measurement. In: M. Kelly (ed) The management of Software Quality. Ashgate Technical/Gower Press, 1993.
5. Spaccapietra S. User Interfaces; Who Cares? In: Proc. of the 20th International Conference on Very Large Data Bases, Santiago, Chile, 1994, p 751.
6. Catarci T., Costabile M.F. (eds.). Special Issue on Visual Query Systems. Journal of Visual Languages and Computing 1995; 6:1.
7. Reisner P. Query Languages. In: M. Helander (ed) Handbook of Human-Computer Interaction. Elsevier Science Publ., 1988, pp 257-280.
8. Yen M.Y., Scamell R.W. A Human Factors Experimental Comparison of SQL and QBE. IEEE Transactions on Software Engineering 1993; 19:4, pp 390-402.

9. Shneiderman B. Improving the Human Factors Aspect of Data Base Interactions. ACM Transactions on Database Systems 1978; 3:4, pp 417-439.
10. Shneiderman B. Software Psychology. Winthrop, Cambridge, Mass, 1980.
11. Thomas J.C. Psychological Issues in Data Base Management. In: Proc. of the 3rd International Conference on Very Large Data Bases, Tokyo, Japan, 1977.
12. Ahlberg C., Williamson C., Shneiderman B. Dynamic Queries for Information Exploration: an Implementation and Evaluation. In: B. Shneiderman (ed) Sparks of Innovation. Human-Computer Interaction, Ablex Publ., Norwood, NJ, 1993, pp 281-294.
13. Greenblatt D., Waxman J. A Study of three Database Query Languages. In: B. Shneiderman (ed) Databases: Improving Usability and Responsiveness, Academic Press, New York, 1978.
14. Boyle J.M., Bury K.F., Evey J.E. Two Studies Evaluating Learning and Use of QBE and SQL. In: Proc. of the 27th Annual Meeting Human Factors Soc., 1983, pp 663-667.
15. Catarci T., Santucci G. Diagrammatic vs Textual Query Languages: A Comparative Experiment. In: Proc. of IFIP W.G. 2.6 Working Conference on Visual Databases, Lausanne, Switzerland, 1995, pp 57-74.
16. Angelaccio M., Catarci T., Santucci G. QBD*: A Fully Visual Query System. Journal on Visual Languages and Computing 1990; 1:2, pp 255-273.
17. Angelaccio M., Catarci T., Santucci G. QBD*: A Graphical Query Language with Recursion. IEEE Transactions on Software Engineering 1990; 16:10, pp 1150-1163.
18. Catarci T., Santucci G. Query By Diagram: A Graphical Environment For Querying Databases. In: Proc. of the ACM SIGMOD Conference on Management of Data, 1994, p 515.
19. Massari A. An Iconic Query System for Large Medical Databases. Ph.D thesis, Dipartimento di Informatica e Sistemistica, University of Rome "La Sapienza", 1995.
20. Massari A., Pavani S., Saladini L., Chrysanthis P.K. QBI: Query By Icons. In: Proceedings of the ACM SIGMOD Conference on Management of Data, 1995.
21. Haber E.M., Ioannidis Y.E., Livny M. OPOSSUM: Desk-Top Schema Management through Customizable Visualizations. In: Proc. of the 21th Int. Conf. on Very Large Databases VLDB'95, 1995, pp 527-538.
22. Chen P.P. The Entity-Relationship Model toward a Unified View of Data. ACM Transactions on Data Base Systems 1976; 1:1, pp 9-36.
23. Massari A., Chrysanthis P.K. Visual Query of Completely Encapsulated Objects. In: Proc. of the Fifth International Workshop on Research Issues on Data Engineering, Taipei, Taiwan, 1995, pp 18-25.
24. Catarci T., Santucci G., Angelaccio M. Fundamental Graphical Primitives for Visual Query Languages. Information Systems 1993; 18:2, pp 75-98.
25. Catarci T., Santucci G. Graphical Primitives for Querying Heterogeneous Databases. In: H.Kangassalo et al. (eds) Information Modelling and Knowledge Bases IV, IOS Press, 1992, pp 106-125 (Proceedings of the Second European-Japanese Seminar on Information Modelling and Knowledge Bases, Finland).
26. Batini C., Catarci T., Costabile M.F., Levialdi S. On Visual Representations for Database Query Systems. In: Proc. of the Second International Conference "Interface to Real & Virtual Worlds", Montpellier, France, 1993, pp 273-283.
27. Santucci G., Sottile P.A. Query By Diagram: a Visual Environment for Querying Databases. Software Practice and Experience 1993; 23: 3, pp 317-340.
28. Chang S.K., Hou T.Y, Hsu A. Smart Image Design for Large Image Databases. Journal of Visual Languages and Computing 1992; 4:3.
29. Massari A., Pavani S., Saladini L. QBI:An Iconic Query System for Inexpert Users. In: Proc. of the Workshop on Advanced VisuaI Interfaces (AVI'94), Bari, Italy, 1994, pp 240-242.
30. Maier D., Ullman J.D. Maximal Objects and the Semantics of Universal Relation Databases. ACM Transactions on Database Systems 1983; 1:8, pp 1-14.
31. Catarci T., Chang S.K., Costabile M.F., Levialdi S., Santucci G. A Visual Interface for Multiparadigmatic Access to Databases. IEEE Transactions on Knowledge and Data Engineering 1996; to appear.
32. Cruz I.F. DOODLE: A Visual Language for Object-Oriented Databases. In: Proc. of the ACM SIGMOD Conf. on Management of Data, 1992, pp 71-80.
33. Jarke M., Vassiliou Y. A Framework for Choosing a Database Query Language. ACM Computing Surveys 1985; 17:3, pp 313-340.
34. Cuff R. N. On Casual Users. International Journal of Man-Machine Studies 1980; 12, pp 163-187.

35. Chandra A.K. Theory of Database Queries. In: Proc. of the ACM Symp. on Principles of Database Systems, 1988, pp 1-9.
36. Kanellakis P.C. Elements of Relational Database Theory. In: J.van Leuween (ed.) Handbook of Theoretical Computer Science, Elsevier Science Pub, 1990, pp 1073-1156.
37. Catarci T. On the Expressive Power of Graphical Query Languages. In: Proc. of the 2nd IFIP W.G. 2.6 Working Conference on Visual Databases, Budapest, 1991, pp 404-414.
38. Codd E.F. Relational completeness of database sub-languages. In: R.Rustin (ed) Data Base Systems, Prentice Hall, Englewood Cliffs, 1972, pp 65-98.
39. Jain R. (ed). Notes from the NSF/ARPA Visual Information Management Workshop. M.I.T. Media Laboratory, Boston, USA, June 1995. (`http://www.virage.com/vir-res`)