Distributed and Parallel Databases 4, 249–269 (1996)

Supporting Mobile Database Access through Query by Icons

ANTONIO MASSARI massari@infokit.dis.uniroma1.it
Dipart. di Informatica e Sistemistica, University of Rome "La Sapienza", 00198—Roma, Italy

SUSAN WEISSMAN suew@cs.pitt.edu
PANOS K. CHRYSANTHIS panos@cs.pitt.edu
Department of Computer Science, University of Pittsburgh, Pittsburgh, PA 15260, U.S.A.

Received May 9, 1995; Accepted February 12, 1996

Recommended by: Daniel Barbara, Ravi Jain and Narayanan Krishnakumar

Abstract. In this paper, we present both the theoretical framework and a prototype of a query processing facility that supports the exploration and query of databases from a mobile computer through the manipulation of icons. Icons are particularly suitable for mobile computing since they can be manipulated without typing. The facility requires no special knowledge of the location or the content of the remote database nor understanding of the details of the database schema. Its iconic query language involves no path specification in composing a query. The query facility provides metaquery tools that assist in the formulation of complete queries in an incremental manner on the mobile computer and without involving access to the actual data in the remote database. By not requiring constant access and caching of the actual data, it is able to effectively cope with the inherent limitations in memory and battery life on the mobile computer, disconnections and restricted communication bandwidth, and the high monetary cost of wireless communication.

Keywords: mobile query processing, iconic query language, mobile computing, semantic distance, semantic and object model, path computations

1. Introduction

Advances in computer and wireless communication technologies have not only affected the way that we compute but, more significantly, they are changing the way we live and do business. For example, mobile users of a hospital paramedic unit arriving at an accident site need to easily access the medical history of the victims, regardless of the location and form of the information. They also need the capability of quickly locating and contacting medical personnel nearest to the accident site. That is, mobile users by means of hand-held computers equipped with a wireless interface, should be able (1) to compose a database query with minimum or no knowledge of how the database is structured and where it is located, and (2) to compose the query with a few key selections and minimum typing [3].

Motivated by these requirements, we present in this paper a query processing facility suitable for mobile database applications. The query processing facility, called *Query By Icons* (QBI), considers the inherent limitations in memory and battery power on the mobile computer, disconnections of the mobile computer for substantial periods, restricted communication bandwidth, and high monetary cost of wireless communication [4, 15].

The salient features of QBI are the following:

- An *iconic visual language* interface, which allows a user to compose a database query by manipulating icons using a pointing device like a light-pen on a hand-held pen-computer. Both structural information and constraints are visualized whereas the implicit ambiguity of iconic representation is resolved by automatically generated natural language.
- A *semantic data model* that captures locally within the mobile computer most of the aspects of the database structures while presenting the user with a set of simple representation structures. That is, a user is not required to have any special knowledge of the content of the underlying database nor the details of the database schema. A user perceives the whole database from any single focal object, as classes of objects expressed as *generalized attributes* of the focal object. Generalized attributes encapsulate and hide from the user the details of specifying a query.
- *Intensional* or *metaquery* tools that assist in the formulation of a complete query during disconnections. A query is formulated in an incremental manner without accessing actual data in the remote database to materialize intermediate steps. Data are accessed and transmitted back to the mobile computer *only* when a complete query is materialized.

While an iconic interface allows fast interactions (faster than typing) even when the user is moving, query formulation using intensional data offsets the expense and limitations of frequent wireless communication that is inherent to extensional browsing systems, e.g., [27, 29]. Metadata can be cached on the disk on the mobile computer since its definition changes rarely and its size is small compared to the actual database. Frequent communication results in slower response time due to the limited bandwidth of wireless links, as well as constant depletion of the computer's battery. Therefore, users can plan in advance to be disconnected from the network in order to save energy and reduce communication costs while actively exploring the database via intensional information on the mobile computer. In addition, users can continue with the formulation of their queries on the mobile computer even when the computer is accidentally disconnected.

In a mobile database environment, we envision QBI being used on both mobile and stationary hosts to query and explore a large distributed database managed by a number of servers on stationary hosts. In the next section, the conceptional model of QBI that defines the mobile user's perception of a database is discussed. Section 3 describes the components of a QBI prototype and illustrates its functionality through its use to query a medical database. QBI's theoretical framework is presented in Section 4 whereas the notion of *generalized attributes* which is central to QBI and its suitability as a query processing facility for mobile users is formally discussed in Section 5. Section 6 describes and evaluates three algorithms for generating generalized attributes on a mobile computer. The paper concludes with a discussion on related work in Section 7 and future work in Section 8.

2. QBI's conceptual data 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. Users are presented

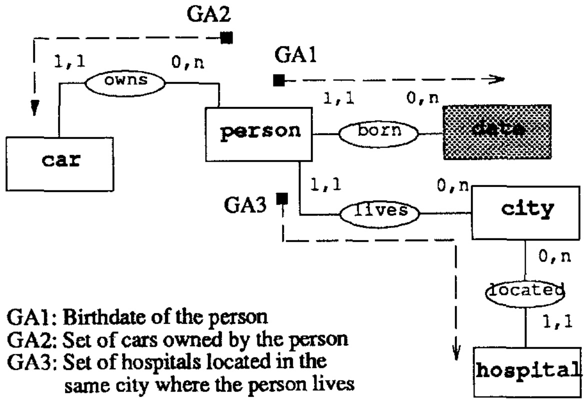

GA2
1,1
0,n
owns
car
GA1
1,1
0,n
person
born
date
GA3
1,1
0,n
lives
city
0,n
located
1,1
hospital
GA1: Birthdate of the person
GA2: Set of cars owned by the person
GA3: Set of hospitals located in the same city where the person lives

Figure 1. Examples of generalized attributes.

with a database abstraction called *complete objects* [25], i.e., completely encapsulated objects, similar to the *universal relation* abstraction in relational databases [23]. 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 an attribute in the ER model [10] represents an elementary property of an entity, a GA expresses a *generic property* of a class.

Further, GAs encapsulate both *implicit* and *explicit* relationships among the objects within each focal object. That is, other object classes are viewed as GAs of the focal object. Thus, in QBI, each focal object provides a view of the whole underlying database from its own viewpoint. Let us illustrate the concept of GAs through an example. Assume the underlying database schema depicted in figure 1 using the Binary Graph Model [9, 25], which also forms QBI's theoretical framework (see Section 4). Here the rectangles denote object classes and ovals convey the interaction among classes.

A QBI user observes that the underlying database contains the same object classes shown in figure 1, namely, `person`, `car`, `city`, and `hospital`, but views the entire structure of the database by means of the GAs of any of these object classes. Of the three GAs of the class `person` shown in figure 1, consider attribute GA3 whose value is a subset of the object class `hospital`. By observing GA3, the user perceives that a `hospital` is located in a `city` and is an attribute of `person`. A generalized attribute with similar meaning exists from the viewpoint of `hospital`. That is, from the view of a `hospital` object class, this GA is a subset of persons corresponding to, `"All the people living in the same city where the hospital is located"`. Also, the same information, could be obtained by observing the GAs of `city`.

3. QBI prototype

In this section, we will describe our QBI prototype and how it can be used to query a Medical Database that includes radiological data from a mobile computer.

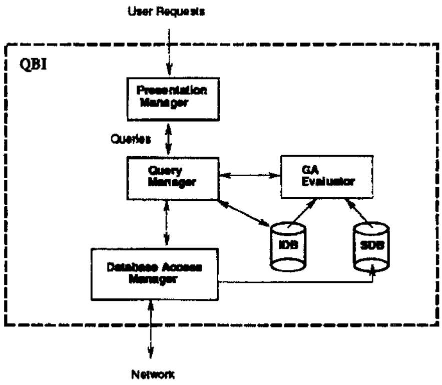


Figure 2. The architecture of QBI.

The QBI prototype is written in C for the MS-Windows environment using the toolkit XVT, and is currently running on NCR System 3125 Pen-top computers with PenDOS and MS-Windows for Pen Computing. The size of the QBI prototype itself is only 0.5 MBytes whereas it stores less than 100 KBytes of intensional information about the radiological medical database.

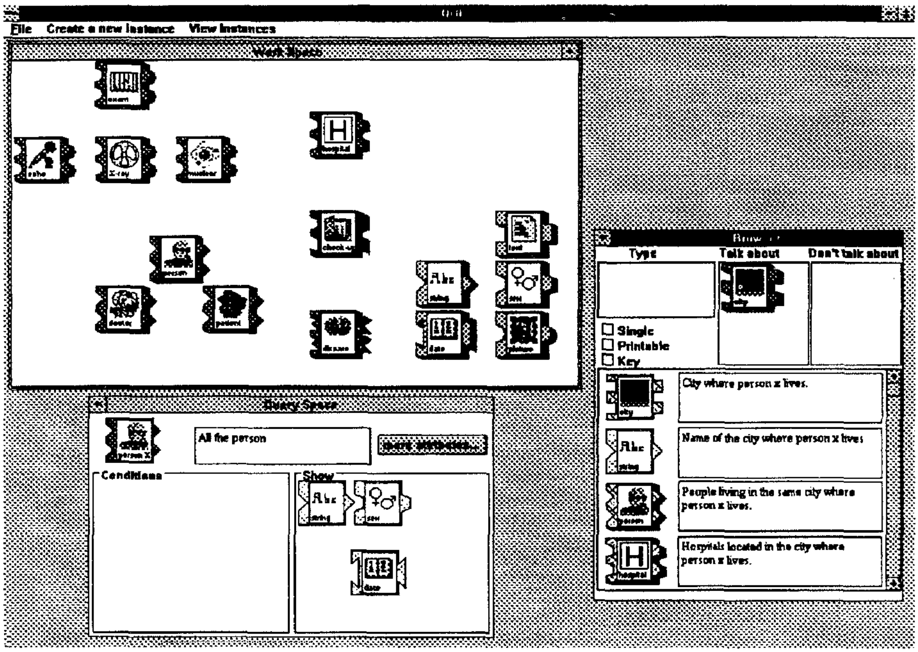
The overall architecture of the QBI prototype is diagrammed in figure 2 and consists of four modules: The *Presentation Manager* which is responsible for all interactions with the user; the *Query Manager* which supports the specification of queries; the *GA Evaluator* which computes the generalized attributes; and the *Database Access Manager* which is responsible for any remote access to the actual data in the database on a stationary host, as well as managing the sporadic updates to the metadata and statistics about the underlying database. GAs are computed on demand because, given an object class, there is potentially an infinite number of GAs associated with an object class and only a small fraction of them is useful in the construction of a particular query.

The execution of the presentation manager as well as of the query manager and the GA evaluator, is supported by two databases, namely, the *Intensional Database* (IDB) and the *Statistical Database* (SDB). The IDB contains all the metadata and the visual data for the iconic representation. Whereas the SDB contains statistical information on the instances in the database used for the evaluation of the GAs.

3.1. QBI's iconic visual language interface

The presentation manager structures the interactions with a user around three windows, each dedicated to a specific aspect in the specifications of a query. The three windows composing the QBI interface are referred to as the *Workspace Window*, the *Query Window* and the *Browser Window*.

3.1.1. Workspace Window. When the QBI application starts, the user is asked to select the database to be considered for querying, in our example a radiological database. In



All the person

□ Single
□ Printable
□ Key

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 3. The QBI interface.

response, the Workspace window appears and displays a set of *icons* corresponding to both *primitive* classes that are actually stored in the database and *derived* classes representing stored queries (see figure 3).

Every icon has an *image* conveying a metaphorical meaning for the class. Below the image is a *label* that allows for easy identification. A full natural language sentence *description* is also provided that can be read by pointing at the icon. This description is automatically generated based on the method described in [8] and is essential for disambiguating the meaning between various icons. Even the *shape* of the icon conveys information. When forming a query, icons representing compatible object classes that are allowed to be combined in a selection condition of a query are associated with a specific geometric outline, similar to a jigsaw puzzle piece. The shape of an icon can appear flat, such as the `patient` icon, or as a *stack* of shapes, such as the `hospital` icon found at the bottom of the Browser Window in figure 3. This stacked representation tells the user how many instances of the object class can be referred to with this one icon. The `hospital` icon appears stacked since it represents a group of hospitals.

3.1.2. Query Space Window. Pointing at an icon in the Workspace corresponds to selecting its object class for a query via the activation of the Query Space. If the class icon for `person` is picked from the Workspace, the Query Space shown at the bottom left of figure 3 will become visible to the user. There are several parts to this window that allow a user to compose a query based on the *select-project paradigm*:

Conditions Space: This space on the left side of the query window allows the user to build both GAs constituting 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: Icons can also be arranged in the section on the right called the *Show Space* which is used for specifying the projection. These icons represent the information the user chooses to view in the output result. An *initial GA set* is displayed (by default) in the Show Space when a class icon is picked and corresponds to the attributes that would appear in an equivalent Entity Relationship representation of the database. For example, the initial GAs of the class `person` are: `Name of the person`, `Sex of the person` and `Birthdate of person`.

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.

3.1.3. The Browser Window and metaquerying. The Browser Window is the interface of the GA generator that allows a user to explore a database by controlling the generation of GAs. From the query space window activated by selecting the icon `person`, a user can see additional GAs not included in the initial GA set displayed by pressing the button labeled `more attributes`. With a given object class, its GAs generation is controlled by a *semantic distance* or *weight* that characterizes, from the viewpoint of an object class, how meaningful a particular GA is for the object class. The additional set of GAs is sorted by their semantic distance so that the most meaningful GAs are shown first. Thus, by observing the top of the list of GAs the user can have an immediate perception of the most meaningful attributes of `person`. The Browser window of figure 3 shows the additional GA `City where person X lives`. Additional GAs such as this one can then be dragged from the Browser window into the Conditions and Show spaces within the Query window when forming a query.

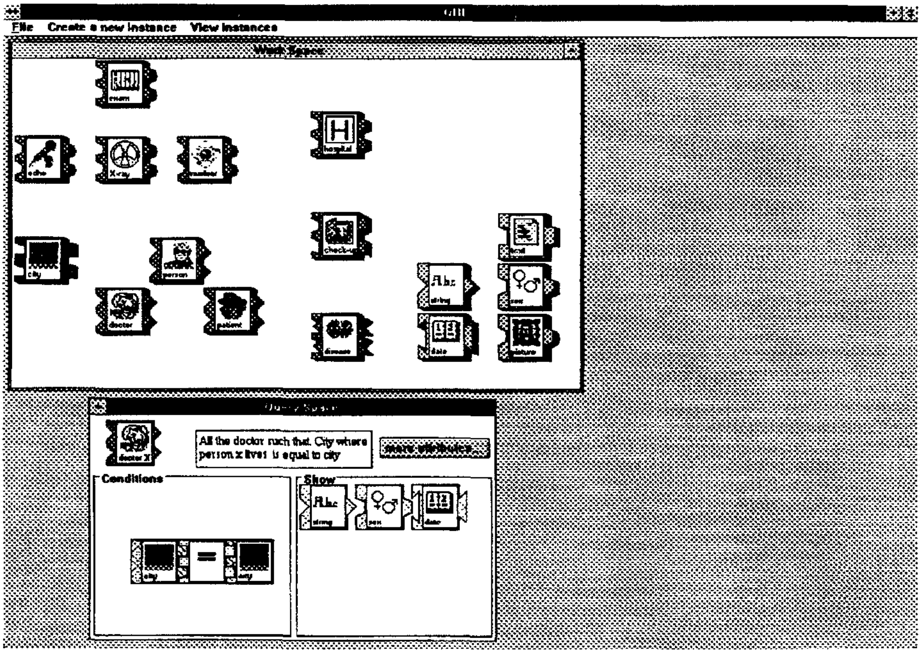
To empower the user with the ability to control his/her view of the database environment, a set of *metaquery tools* are provided within the Browser. These *metaquery operators* permit the specification of filter conditions on the GA set. Hence, a user interested in very distant properties of the class person can easily explore these properties, by restricting the search of the desired GAs within a smaller GA set. For example, one useful metaquery operator is used when a user desires GAs which are associated with a specific object class. If the user is interested in all the GAs that *talk about* city, the icon for `city` from the Workspace window can be moved into the **Talk about** space of the Browser (see figure 3). In particular it is possible to express the following metaquery conditions: (1) *single*, *printable*, *or key* selects only single valued, printable GAs, or key GAs used to identify an instance of a class, respectively, (2) *type* selects all the GAs that represent a subset of a particular object class and, (3) *talk about*, *don't talk about* selects GAs that are associated or not associated with a specified class. All the metaquery conditions are combined in a conjunctive expression by default.

3.2. Query examples

Let us revisit the hospital paramedic unit example mentioned in the introduction. As a patient is rushed to the most appropriate hospital, a specialist living within close proximity of the hospital is notified. For this type of information, we need to determine 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` into the **Type** space of the Browser window. 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 (figure 4). For the projection, the user does not have to pick the name, sex and birthdate of the doctor because they are part of the initial GA set of `doctor` and must be already in the Show space. In order to know in which city the hospital and



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

Figure 4. A query.

doctor are located, the user simply drags the appropriate, i.e., city, icon from the Browser into the Show space.

The result of a query constitutes a derived class of the picked class and as such it can be stored in the intensional database. In the above example, after the user chooses a label, say `Lucky Drs`, for the newly created derived class, the new class is assigned an icon which becomes part of the icon set contained in the workspace window and can be used as if it were a primitive one. The image of this icon will be that of a doctor since this GA is a subset of the object class `doctor`. To materialize this query, all the user has to do is drag this new icon to a special system icon called the "printer". This operation corresponds to forwarding the corresponding query to the remote database, requesting for its execution, and fetching the query result for display.

As seen with this example, QBI is very useful in a mobile computing environment as queries require very little typing. Also, only a small screen is required since queries do not require any form of path specification. The underlying, formal schema is hidden and feedback in the form of natural language and shapes is very helpful for users with a limited knowledge of database languages. In addition, intensional data and metaquery tools are provided to users to allow them to formulate queries even when the computer is disconnected. In the following section, we will formally define the semantic data model used by QBI, as well as the concepts of generalized attributes and semantic distance. In further sections, we will show how the internal algorithms for GA generation were improved for a mobile environment.

4. QBI's theoretical framework

Internally, QBI uses the Binary Graph Model (BGM) [9, 25], a semantic data model, for capturing most of the aspects of the structure of the remote database. The major constructs of this model are: the class of objects, the binary relationship among classes, the ISA relationship between a class and its superclass, and cardinality constraints for the participation of class instances into the relationships. A BGM schema can be expressed as a labeled graph called *typed-graph.*

Definition 4.1 (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 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 $\langle g, c, m\rangle$, 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 4.2 (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 $\langle \mathrm{lbl}_1(n_r)\colon x_1, \mathrm{lbl}_2(n_r)\colon x_2\rangle$, where $\mathrm{lbl}_1, \mathrm{lbl}_2$ are functions returning the labels of the two edges connected to n_r ($\mathrm{lbl}_1, \mathrm{lbl}_2\colon N_R \rightarrow L$) and $\langle x_1, x_2\rangle \in m(n_{c1}) \times m(n_{c2})$ 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. The set of constraints on the database referred to in this paper are the minimum (ATLEAST) and maximum (ATMOST) cardinality constraints, and the subclass-superclass relationship constraint (ISA).

Definition 4.3 (Constraints). The set c contains: (1) ATLEAST(k, n_{c1}, n_r) specifies that an instance of class node n_{c1} can participate in at least k interpretations involving the adjacent role node n_r; (2) ATMOST(k, n_{c1}, n_r) specifies that an instance of class node n_{c1} can participate in at most k interpretations involving the adjacent role node n_r; (3) ISA$(n_{\hat{c}}, n_c)$ specifies that the class $n_{\hat{c}}$ is a subset of the class n_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.* 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 5 shows an example of a typed graph which is a fragment of the medical database used in the previous section. 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.

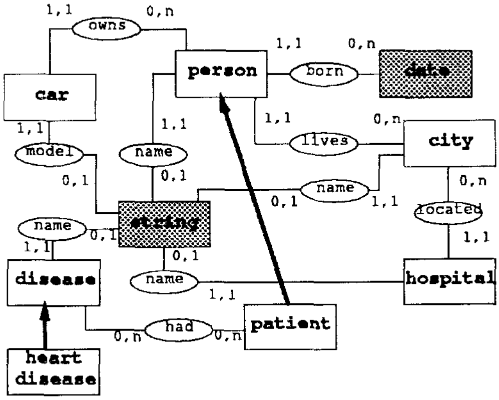


Figure 5. A typed graph.

5. Generalized attributes

As mentioned above, the concept of GA in QBI represents the way in which a user perceives the relationships among objects. Internally, a GA is strictly related to the concept of *path* in a typed-graph capturing the database schema, where a path is a sequence of adjacent class and role nodes always starting and ending with a class node.

Definition 5.1 (Path). Let $\mathcal{G}$ be a typed-graph. A *step* s on $\mathcal{G}$ is the triple $\langle class_1(s), role(s), class_2(s)\rangle$ 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 $s_1, s_2, \ldots, sk$ of steps on $\mathcal{G}$ such that, for $i = 1, 2, \ldots, k-1$, $class_2(s_i) = class_1(s_{i+1})$. The first and the last class node of a path p, i.e., $class_1(s_1(p))$ and $class_2(s_{\text{length}(p)}(p))$, will be denoted with *first*(p) and *last*(p) respectively.

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

Definition 5.2 (Generalized 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).

A path in a typed-graph can be cyclic. Cyclic paths are allowed since they can represent useful properties, e.g., `People living in the same city where the person lives`. As a consequence the set of possible GAs associated with a class node is infinite. 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. A similar notion to that of semantic distance function in QBI is the notion of semantic length in the generation of complete queries from incomplete path expressions [17].

Definition 5.3 (Semantics distance function). Let γ_p be a GA of a class node n_c. The function Semd: $\Gamma(n_c) \rightarrow \Re$ maps γ_p to a real value Semd(γ_p) that represents how much γ_p is semantically distant from n_c.

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. The statistical information can be used to compute the *entropy* of a GA which is the measure of uncertainty in the information theory [13]. The entropy clearly captures the fact that a user considers a GA only if the GA conveys some information. By taking entropy into consideration, a large number of less meaningful GAs, such as "all the persons that have a name equal to the model name of a car", can be discarded.

Given a semantic distance function *Semd* and a *threshold* value $\tau \in \Re$, the finite GA set of n_c, with respect to Semd, will be determined by:

$$\bar{\Gamma}_{\text{Semd}}(n_c) = \{\gamma_p \in \Gamma(n_c) \mid \text{Semd}(\gamma_p) < \tau\}$$

provided that Semd is monotonically increasing.

The following function was implemented in the QBI prototype to compute the semantic distance of a new GA $\gamma_{p'}$ resulting from adding a new step to an existing GA γ_p. Let the new step being added to γ_p be $e = \langle u, v, w \rangle$ (i.e., $p' = p \cup e$).

$$\begin{aligned}\text{Semd}\gamma_{p'} = {} & (c_1 * \text{length}(p')) + (c_2 * \text{num\_cycles}(p') \\ & + (c_3 * \text{max\_cardinality}(p) * \text{atmost\_cardinality}(e) - 1) + \text{NPW} * c_4 \\ & + \mathcal{E} * (c_5/\text{avg\_cardinality}(p') - c_5) + \text{NPU} * c_6\end{aligned}$$

where NPW, NPU and $\mathcal{E}$ are binary flags with 0 or 1 value, and c_1, c_2, c_3, c_4, c_5 and c_6 are positive real constants used as semantic penalties for properties that may exist in a new GA path. As a GA's path becomes longer (length(p') or cyclic (num_cycles(p')), or its cardinality dramatically increases (max_cardinality(p) $*$ atmost_cardinality(e)), so does the semantic distance associated with this GA. The length is defined to be the number of role nodes connectors used by the path and does not include any class-superclass connectors (i.e., "`is a`" connectors).

A path that ends with a class node w which is *not* printable, represents a relationship between n_c and another abstract object class. This object class will contain simple attributes that have not been discovered. However, because there is no information for the user that is directly obtainable from this path, the path is penalized by assigning NPW to 1. Otherwise NPW is 0.

In the QBI prototype, the current weight increase c_6 for going through printable nodes is 300 and is equivalent to the penalty c_2 associated with a class node being a member of a cycle. The other penalty constants are $c_1 = 200$, $c_3 = 0.02$, $c_4 = 20$ and $c_5 = 100$. All penalty constants have be adjusted for better performance after a series of experiments while making sure that the Semd remains monotonically increasing.

6. Mobile GA generation

In a mobile environment, QBI's method of query formulation and use of intensional data limits the cost of frequent wireless communication with respect to the materialization of complete queries. Visualization of the database as well as cost effective query formulation is done primarily through the manipulation of generalized attributes. As stated above, a GA γ_p is simply a path p in a typed-graph $\mathcal{G}$. Every time a step is added to an existing GA path, a new GA is formed and its associated (real number) semantic distance value (sd-value) is computed by the function Semdγ_p. The generation of GAs is an instance of a *path computation* [2, 16], and is clearly the most computationally expensive part of

QBI. The cost of the semantic distance function depends on its complexity which, in turn, is a measure of its accuracy to express the meaningfulness of a GA. However, this cost is independent of the environment in which QBI is executing. On the other hand, the value for the threshold that terminates the generation of GAs can be tuned to consider the capabilities of the system. In the case of the mobile computer, the threshold τ is defined as a function of the available memory, the energy level and the response time:

$$\begin{aligned}\tau = {} & C_1(\text{free\_memory}) + C_2(\text{energy\_level}) + C_3(\text{cpu\_speed}) \\ & + C_4(\text{resp\_requirements})\end{aligned}$$

where C_i are user defined parameters. Given that the first two parameters, free_memory and energy_level, vary over time, the threshold τ dynamically changes as well.

The initial GA evaluator in our QBI prototype traverses the typed-graph in a *depth-first search* (DFS) manner. Although this is also the approach traditionally used in path computations [2, 16], it has turned out to be unsuitable for mobile operations. First, since it does not generate GAs sorted based on their sd-value, an additional sorting phase is required for the presentation. More importantly, the DFS strategy is not compatible with a dynamically defined threshold. DFS allows for the possibility of a dynamically set threshold to terminate the execution of the GA evaluator before the generation of GAs associated with a low sd-value and after spending a significant amount of time in generating GAs with higher sd-values. For these reasons, we have explored two other alternatives, the *best-first search* (BEST) and *breadth-first search* (BFS) based GA evaluators.

The BEST algorithm explores the given graph $\mathcal{G}$ by maintaining an order among the GAs found based on the semantic distances associated with the GAs. At each iteration, BEST always considers the GA with the minimal semantic value. The BEST-based algorithm uses: (1) a sorted List to maintain an order among the GAs produced, (2) (n_c), the starting class node chosen by the user, (3) a path p associated with every GA γ_p and a total semantic distance Semd(γ_p), and (4) the function weight (u, w), which computes the distance between two class nodes u and w, where $\mathcal{G}$ is the given typed-graph and for each step $e \in \mathcal{G}$, $e = \langle u, v, w \rangle$. Below are shown the basic steps for the algorithm.

```
BEST ($\omega = n_c, \gamma_p, \tau, \Gamma(n_c)$)
Do
  For each step $e = \langle u, v, w \rangle$ where $u = \omega$
           Mark every class node in $p$ of $\gamma_p$ as newly visited in order to detect cycles.
           Calculate $\text{Semd}(\gamma_{p'}) = \text{Semd}(\gamma_p) + \text{weight}(u, w)$
           If $(\text{Semd}(\gamma_{p'})) \leq \tau$ Then
                     $p' = p \cup e$
                     Sorted_Insert(List, $\gamma_{p'}, p', \text{Semd}(\gamma_{p'})$)
  End For

  If the List is not empty
           $\gamma_p$ = First_Of_List(List)
           $\omega = last(p)$   (the $p$ associated with our new $\gamma_p$)
           $\Gamma(n_c) = \Gamma(n_c) \cup \gamma_p$
While the List is not empty
```

The BFS algorithm, the second alternative GA evaluator, explores the given typed-graph $\mathcal{G}$ in a level-by-level fashion. Only when all the class nodes at a given level are explored does the algorithm move on to the next level. All of the items used by BEST are also used by BFS, except that maintaining an order among the GA paths that are to be expanded is now done by a Queue instead of a List.

In accordance to the basic BFS algorithm, the Queue was sorted every time all the class nodes at a level x were explored. Although, sorting would be performed more often, a very small number of GAs would be sorted each time, and BEST's incremental sort with many comparisons would be avoided. However, it has turned out that this is not enough, since there are *no* order guarantees among the weights of GAs ending at different levels along different paths. Because of the binary flags NPW, NPU, and $\mathcal{E}$ whose values are dependent on the type of the class node, all paths produced by p that end at level x are *not* guaranteed to have smaller sd-values than all paths found at level $x + 1$. Hence, BFS requires an explicit sorting phase as DFS.

6.1. Advantages and disadvantages of DFS, BFS, and BEST

With BEST and BFS, the main advantage over DFS is efficiency in finding meaningful GAs. BEST generates GAs in the order of their meaningfulness to the user based on their semantic distance from the focal object class n_c. With BFS, GAs with the shortest paths are generated first, and it is highly likely that these paths are very semantically meaningful from the viewpoint of n_c due to the monotonically increasing property of Semd along a path. The ordering performed by BEST and BFS is useful in a mobile environment since the semantic distance threshold τ could be set by a function that describes the limitations of the mobile unit. With BEST, only the most meaningful GAs with respect to these limitations would be generated, and it does not require a separate explicit sorting phase.

However, unlike DFS, both BEST and BFS maintain complex data structures in the form of a sorted List or Queue. In the worst case, each element added to the List must be compared to all the other elements before finding its correct location. If the graph is very dense, with each node ω having a large degree, the number of comparisons will be very high. An additional indexing structure could be used to combat the cost of insertion. This approach, however, would require more space. One advantage BFS has in this regard is that it does not need to perform an expensive incremental sorting on each insertion of a new GA the way BEST does.

In addition, adding a step $e = \langle u, v, w \rangle$ to an existing path p may cause the ending class node w to become a member of a cycle. Detection of this in DFS only requires a marker associated with the node w. If the class node w is already a part of the path p before the addition of e, then the node will be marked. Hence, another factor that increases the execution time of BEST and BFS when compared to DFS, is the method needed for cycle detection. In order to detect if the node w is part of a cycle, all the class nodes of the path p currently being expanded must be marked. This prevents the detection of a false cycle whenever two separate routes from the focal point object n_c are being expanded concurrently and both reach the same node. However, the cost for this is one traversal of the path p every time a step is added. Since each of these algorithms have comparable advantages and disadvantages, further investigation is necessary in order to determine which one is the most useful with respect to a mobile computing environment.

6.2. *Evaluating the GA generation methods*

To the user, the two most important criteria are *response-time* and *quality* of GAs i.e., the semantic distances of the GAs produced. These are two important criteria in determining how well each of the algorithms proposed for the GA evaluator perform. However, within a mobile environment, the GA evaluator should operate without depleting a large amount of the mobile unit's resources. The method proposed above for controlling the amount of resources used by the GA evaluator is the notion of a dynamically changing threshold τ. How effectively a changing threshold performs with respect to finding meaningful attributes and response-time must be evaluated.

Two tests were done in order to evaluate the three GA Generation Methods discussed in the previous sections. The first test compares the three algorithms with regards to the quality of attributes found by each. The second test evaluates the three implementations under a dynamically changing threshold. These experiments were done using an Intel 486DX, 66 MHz PC with 16 MBytes RAM.

Test 1: Meaningful attribute test (MAT)

Task: Each algorithm was given the task of finding a given number (X) of GAs for the `doctor` object class in the radiological database of the QBI prototype.

Parameters: The semantic weight threshold *remained* a constant 1800. At this value, the system produced a sorted attribute list (SAL) of 947 GAs. For each successive test run, the number of GAs required (X) was *increased* by 100.

Each algorithm was timed from the moment it was invoked until it was able to produce a sorted list of the required number (X) of attributes. In addition, to measure the quality of the attributes produced by the algorithm, a comparison was made to see how many of the attributes produced by the algorithm matched the first X attributes found in SAL. Figure 6 is a graph of these test runs.

Test 2: Dynamic threshold test (DTT)

Task: Each algorithm was required to find all the GAs below a given semantic distance threshold τ for the `doctor` object class in the radiological database of the QBI prototype.

Parameters: The semantic distance threshold τ was changed for each successive test run. It was incremented by a value of 100 for each run.

Each algorithm was timed from the moment it was invoked until it was able to produce a sorted list of attributes below the given semantic distance threshold. Figure 7 is a graph that

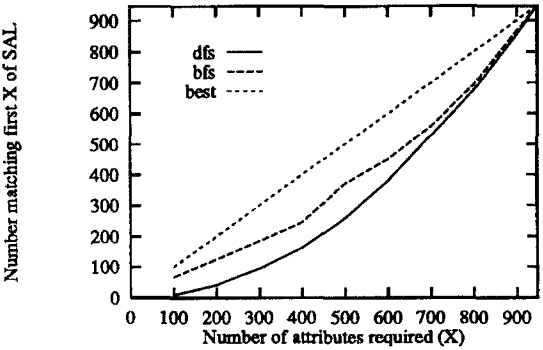


Figure 6. MAT: Quality of attributes found.

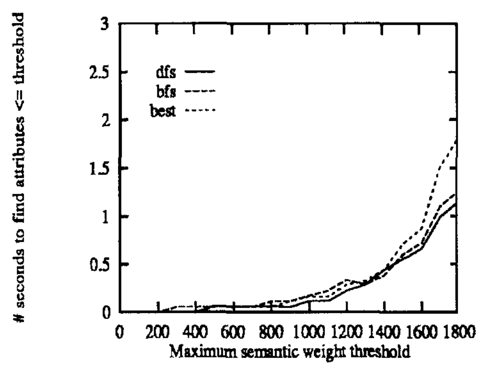


Figure 7. DTT: Time to find attributes below a given threshold value.

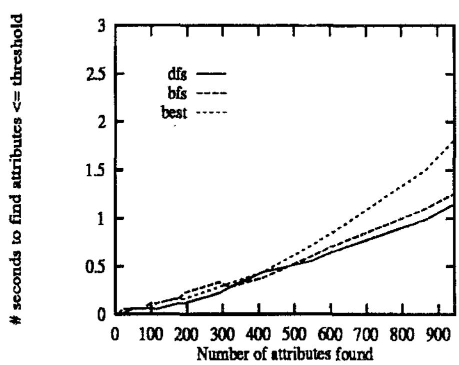


Figure 8. DTT: Number of attributes found below a given threshold.

shows how many seconds it took each algorithm to find all the GAs below a given threshold τ. For each threshold τ, a certain number of attributes in the typed-graph $\mathcal{G}$ will have distances below τ. In order to compare the results of MAT with this test, another graph shown in figure 8 was created. In this graph, the amount of time required by each algorithm is mapped to the number of attributes within $\mathcal{G}$ that are below the corresponding threshold.

6.2.1. Implications of MAT and DTT. Suppose that in a mobile application users would be able to obtain all the information they wanted if the Browser Window always displayed the first 450 most meaningful attributes for the `doctor` object class. Figure 6 shows that BFS finds the top 450 most meaningful attributes by generating a total of 600 attributes from the typed-graph $\mathcal{G}$. From figure 7, it is apparent that BFS finds these 600 attributes within 0.72 seconds. Also, from figure 8, finding 600 attributes within 0.72 seconds using BFS corresponds to a threshold of $\tau = 1600$. Of course, these figures could have been calculated using any of the three implementations. Below is a table that corresponds to the answers for DFS, BFS, and BEST for the Browser Window to show 400, 600, and 800 of the most meaningful attributes to the user.

Number attributes	τ DFS	sec.	τ BFS	sec.	τ BEST	sec.
400	≈1620	≈.7	≈1500	≈.6	≈1390	≈.42
600	≈1640	≈.8	≈1630	≈.8	≈1590	≈.85
800	≈1725	≈1.0	≈1680	≈1.0	≈1660	≈1.3

MAT and especially figure 6, show that BEST finds GAs in order of increasing semantic distance while DFS performs the worst at generating meaningful GAs. This result was expected since BEST does not require an additional explicit sorting phase. BFS's performance was between DFS and BEST. The algorithm does require an explicit sorting phase, but because it generates the GAs by levels, its performance in terms of generating meaningful GAs was better than DFS.

In figure 7, we observe that as the threshold changes, so does the amount of time each algorithm requires to do its work. This happens because as the threshold increases, more of the typed-graph is explored. All three algorithms take approximately the same amount of time until the semantic weight threshold reaches 1400. Soon after this point, DFS begins to take the least amount of time and BEST the most. This phenomenon can be explained by noticing that as the exploration of the graph moves further away from the focal point object class n_c, more attributes are generated that have the same properties and, therefore, approximately the same semantic distance. These large, similar groups of GAs are far enough away from n_c to have approximately the same semantic meaning from the viewpoint of n_c. Since BEST must maintain a sorted list, every time a large group of closely weighted GAs are produced, they must be placed in their proper position in the list. This requires a large number of traversals and accounts for the decrease in BEST's performance as the threshold τ increases.

6.2.2. An integrated mobile GA evaluator. Within a mobile environment, the mobile computer could set the semantic distance threshold by a function depending on how much memory, processing, battery power, and delay the host and user can afford. Therefore, the mobile computer would be trying to reserve its resources and not waste time or energy finding GAs that are of little value to the user. BEST can generate the most meaningful GAs with respect to these limitations. Although the BEST algorithm is more compatible with a mobile environment, its response-time suffers as the threshold is dynamically increased.

Although our original intention was to replace the currently used DFS algorithm in QBI with either the BEST or BFS algorithms, our experimentation clearly showed that one algorithm does not meet all the needs of a mobile GA evaluator. Therefore, we propose to combine these techniques into one integrated mobile GA evaluator. Since each algorithm only requires a few kilobytes, this proposed integrated evaluator will not require a dramatic increase in the amount of code kept on the mobile computer. BEST will be the default algorithm of the mobile GA evaluator, since it does produce a sorted list of GAs. In a mobile environment, as long as a low semantic threshold τ is computed, the BEST algorithm will be used. However, as the limitations of the mobile host are relaxed, the GA evaluator will switch to a BFS that can facilitate a broader, sweeping search of the graph with a better response-time. Finally, whenever there are few restrictions on the mobile host (e.g., when the mobile host is stationary and attached to docking mobility-support station), or the user wants to examine a large number of GAs, a switch to a "focus" DFS algorithm would facilitate the exploration of the database using more resources. Therefore, GA evaluator thresholds can be set in much the same way that the ER threshold was set for the query window (see Subsection 3.12). As τ changes dynamically and captures the status of the mobile host, the search technique used by the GA evaluator will also change to accommodate these limitations. We are currently investigating this integrated mobile GA evaluator.

7. Related work

Most of the work related to our approach has been done in the areas of database graphical user interfaces, and data modeling.

In the area of graphical user interfaces a large amount of research has been produced with the purpose of facilitating user interaction while still maintaining the highly expressive power of the query language [7, 20]. Most of the proposed systems adopt form, tabular, or diagram based visual paradigms. Early examples of these types of visual query languages that use a *relational external data model* are QBE [35] and G+ [11]. In QBE, the query is made by filling in templates of relations. Users do not need to remember attribute names or variable names. Queries are specified by typing example tuples expressing the information that is being requested. G+ makes use of a diagrammatic paradigm by using a graph whose edges correspond to the tuples in a relation.

Semantic data models go even further than the relational model in terms of providing the user with a more abstract logical view of the data. Of these, the ER model [10] is often used as the external data model in existing visual query systems. GORDAS [12], QBD* [5], GRAQULA [28, 33], and GQL/ER [34] are examples of graphical visual query systems that provide the user with an ER diagram of the schema. Queries in these systems are formulated by drawing nodes and edges to be matched in the schema diagram. That is, queries are specified as subgraphs of the ER schema diagram with certain nodes and edges replicated as necessary. Selection conditions and projections are specified as annotations of the nodes and edges. For example, in GORDAS and QBD*, once a user selects the entities and relationships of interest, a simplified hierarchical diagram of the schema is provided in order to aid in the formulation of queries. In general, the difference between these systems is their varying support in the specification of aggregation, quantification, and recursion. Further, GQL/ER combines features of the universal relational model and the ER model without the support of aggregation or quantification. PICASSO [19], on the other hand, provides an external universal relation data model that interfaces a universal relational database system. In PICASSO, maximal objects are represented as hyperedges in a hypergraph which contains textual attribute labels. Queries are formulated via mouse clicks which reveal pop-up menus that allow for the selection of aggregate and set operators as well as comparison operators used in predicate formulation. Adding a hyperedge with the mouse creates a tuple variable and, therefore, no character-type tuple variables are necessary. Another visual query system which as QBI is based on a richer semantic model than the ER model and universal relational model is Ski [21]. In Ski, by means of a set of semantics operators, users can dynamically construct portions of the database schema and peruse the schema for related information, having complete access to an underlying semantic data language. Similar to QBI, this perusal is not performed navigationally but semantically. As opposed to QBI, Ski is diagram based and supports navigation through the paths of the underlying database schema.

Compared to diagrammatic visual languages, the ease and effectiveness of QBI with respect to unsophisticated users was established through an empirical evaluation of QBI and QBD* [24, 6]. In this study users were classified into unskilled users with little, if any, training in databases, and skilled ones. The two performance measures used were the time in seconds to complete a query and accuracy of the query. Each group of users after a short training session of equal times in using QBI and QBD* were given six queries of different levels of complexity in natural language. Users were given these queries in different orders in order to minimize the learning effect. In general, unskilled users did better with QBI whereas skilled ones felt more comfortable using QBD*, particularly in expressing queries characterized by a high semantic distance value involving paths of length 4, or more, and with no cycles. The reason was that skilled users perceived the whole path not as a single complex function, i.e., GA, but as a sequence of steps that can be manually built and controlled. On the other hand, there was a significant difference in accuracy and performance for queries with low semantic distance value or queries involving cycles. In the presence of cycles, QBD* users get much more confused because they see multiple copies of the same form, each corresponding to a different occurrence of the same concept (entity or relationship). 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 of the same concept.

When compared to the work done involving icon based visual paradigms, it is evident that a greater amount of work has been performed using form and diagrammatic paradigms. However, the small screen space of the typical notebook or palmtop computer and the limited possibility of using a keyboard, make the iconic approach particularly suitable for the users of a mobile system. In general, the main difference between QBI and the other iconic interfaces proposed in the literature [14, 29, 30] is in the way icons are defined and used for expressing concepts. In particular, other systems do not usually assign uniform semantics to icons. Also, as opposed to QBI, these systems adopt the extensional browsing approach (that is, browsing of instances in the remote database) as the principal querying strategy [27, 29] hence making them unsuitable for mobile environments that are characterized by low communication bandwidth over expensive wireless communication links.

All visual query interfaces discussed above have been proposed in the context of workstations with large screens, graphics capabilities and pointing devices. The need for an alternative visual query paradigm for mobile, pen-based computers that takes into consideration the requirements of mobile users, such as exploration of a large database schema, and the limitations of mobile computers, such as small screen and no keyboard, was first identified in [3]. As opposed to QBI, the proposed alternative is form-based whereas the external data model is a multi-level semantic data model which uses the universal relation approach at different levels to coalesce related information and eliminate low-level information not relevant to the user. As stated earlier, the concept of GAs in QBI serves a similar purpose, coalescing related information of an object from the perspective of the user and representing it on the screen with an icon.

Recognizing that query languages which require fully specified paths are too restrictive, a number of authors have proposed various solutions. In [18], *path expressions* are examined and form the basis of the XSQL system. XSQL allows the specification of *path variables* by means of which incomplete path expressions can be specified. In [17], path expressions are considered to be abbreviated queries within a user interface to a database system. Given an ambiguous path expression which could result in multiple possible paths, the task is to find those completions most likely intended by the user.

The idea of the Universal Relation Model [22, 23, 31] is that access paths are embedded in attribute names and for every set of attributes X there is a unique basic relation that the user has in mind. This relation is computed through the *Window Function* on the set of attribute names X. Within a Window Function a *decision problem* concerning which is the most meaningful attribute is tackled. The choice of assigning a meaning to an attribute name is based on the analysis of the schema of the underlying database and various kinds of dependencies. With this approach, the same attribute name can have different meanings if used in different contexts; as a consequence, even if the user is not required to know the internal schema of the database, she/he must be aware of the domain of interest.

The idea of presenting to the user a simplified structure of the database by evaluating the semantics of the attributes, is common to both QBI and the Universal Relation approach. However, the querying strategy, is slightly different. Instead of assigning a meaning to an attribute *after* the query has been specified, QBI uses the semantic distance function to present to the user all the meaningful attributes *before* the query is composed. Moreover, the use of a semantic model and statistical information on the database extension allows the definition of a richer notion of meaningfulness of an attribute.

8. Conclusions

In this paper, we have described an icon-based query processing facility called QBI, suitable for mobile users. That is, QBI satisfies all three of the criteria identified in the introduction for an effective mobile query processing facility:

(1) QBI allows the construction of a database query with no special knowledge of how the database is structured and where it is located. Its iconic visual query language does not involve path specification in composing a query. Thus, it is equally useful to both unsophisticated and expert mobile users.

(2) Users primarily interact with the system with a pointing device, such as a pen or a mouse, and compose a query by arranging icons. Thus, it overcomes any size limitations of a mobile computer while new requirements are not imposed.

(3) QBI's algorithms, particularly the metaquery tools and GA evaluator, are designed to effectively operate under limited memory and disk capacity, limited battery power, and restricted wireless communication bandwidth.

As mentioned above, we are looking into integrated path computations under resource constraints suitable for mobile query processing. Further, we are interested in extending aspects of this work in order to minimize the amount of retrieved and transmitted data over wireless links.

Acknowledgments

This work was supported in part by National Science Foundation under grants IRI-9210588 and IRI-95020091 (USA), Integra Sistemi Interattivi, and Computer and Microimage S.p.A. (Italy). We thank S. Pavani and L. Saladini for their participation in the development of the prototype.

References

1. S. Abiteboul and A. Bonner, "Objects and views," Proceedings of the Int'l Conference ACM-SIGMOD, Denver, Colorado, June 1991, pp. 238–247.
2. R. Agrawal, S. Dar, and H. Jagadish, "Direct transitive closure algorithms: Design and performance evaluation," ACM Transaction on Database Systems, vol. 15, no. 3, 1990, pp. 427–458.
3. R. Alonso, E. Haber, and H. Korth, "A database interface for mobile computers," Proceedings of the 1992 Globecom Workshop on Networking of Personal Communication Applications, Dec. 1992.
4. R. Alonso and H. Korth, "Database issues in nomadic computing," Proceedings of ACM SIGMOD Int'l Conference on Management of Data, May 1993, pp. 388–392.
5. M. Angelaccio, T. Catarci, and G. Santucci, "QBD*: A graphical query language with recursion," IEEE Transactions on Software Engineering, vol. 16, no. 10, 1990, pp. 1150–1163.
6. A.N. Badre, T. Catarci, A. Massari, and G. Santucci, "Comparative effectiveness of a diagrammatic vs. an iconic query language," (submitted for publication), Feb. 1995.
7. C. Batini, 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", Mar. 1991.
8. G. Bono and P. Ficorilli, "Natural language restatement of queries expressed in a graphical language," Proceedings of the 11th Int'l Conference on Entity-Relationship Approach, Germany, Oct. 1992, pp. 357–374.
9. T. Catarci and G. Santucci, "Fundamental graphical primitives for visual qery languages," Information Systems, vol. 3, no. 18, 1993, pp. 75–98.
10. P.P. Chen, "The entity relationship model toward a unified view of data," ACM Transactions on Database Systems, vol. 1, no. 1, 1976.
11. I.F. Cruz, A.O. Mendelzon, and P.T. Wood, "G+: Recursive queries without recursion," Proceedings of the 2nd Int'l Conference on Expert Database Systems, 1988, pp. 355–368.
12. R. Elmasri and G. Wiederhold, "GORDAS: A formal high-level query language for the entity-relationship model," Proceedings of the 2nd Int'l Conference on Entity-Relationship Approach, Washington, D.C., 1981, pp. 49–72.
13. R.G. Gallager, Information Theory and Reliable Communication, Wiley: New York, 1968.
14. I.P. Groette and E.G. Nillson, "SICON: An icon presentation module for an E-R database," Proceedings of the 7th Int'l Conference on Entity Relationship Approach, Roma, Italy, 1988, pp. 271–289.
15. T. Imielinski and B.R. Badrinath, "Mobile wireless computing: Challenges in data management," Communication of ACM, vol. 37, no. 10, 1994, pp. 18–28.
16. Y.E. Ioannidis, R. Ramakrishnan, and L. Winger, "Transitive closure algorithms based on graph traversal," ACM Transactions on Database Systems, vol. 18, no. 3, 1993, pp. 512–576.

17. Y.E. Ioannidis and Y. Lashkari, "Incomplete path expressions and their disambiguation," Proceedings of the ACM SIGMOD Int'l Conference on Management of Data, Minneapolis, MI, May 1994, pp. 138–149.
18. M. Kifer, W. Kim, and Y. Sagiv, "Querying object oriented databases," Proceedings of the ACM SIGMOD Int'l Conference on Management of Data, May 1992, pp. 138–149.
19. H. Kim, H. Korth, and A. Silberschatz, "PICASSO: A graphical query language," Software Practice and Experience, vol. 18, no. 3, 1988, pp. 169–203.
20. W. Kim, Introduction to Object-Oriented Databases, MIT Press: Cambridge, MA, 1990.
21. R. King and S. Melville, "Ski: A semantics-knowledgeable interface," Proceedings of the 10th Int'l Conference on Very Large Data Bases, Singapore, Aug. 1984, pp. 30–33.
22. D. Maier, D. Rozenshtein, and D.S. Warren, "Window functions," Advances in Computing Research, vol. 3, 1986, pp. 213–246.
23. D. Maier and J.D. Ullman, "Maximal objects and the semantics of universal relation databases," ACM Transactions on Database Systems, vol. 1, no. 8, 1983, pp. 1–14.
24. A. Massari, "An icon based query system for radiological data," Ph.D. Thesis, Dipartimento di Informatica e Sistemistica Universita' di Roma "La Sapienza," Nov. 1995.
25. A. Massari and P.K. Chrysanthis, "Visual query of completely encapsulated objects," Proceedings of the 5th Int'l Workshop on Research Issues in Data Engineering-Distributed Object Management, Taipei, Taiwan, March 1995, pp. 18–25.
26. A. Massari, S. Pavani, and L. Saladini, "QBI: An iconic query system for inexpert users," Proceedings of the Workshop on Advanced Visual Interfaces, Bari, Italy, June 1994, pp. 240–242.
27. A. Motro, A.D. Atri, and L. Tarantino, "KIVIEW: The design of an object oriented browser," Proceedings of the 2nd Conference on Expert Database Systems, Virginia, 1988, pp. 107–131.
28. G.H. Sockut, L.M. Burns, A. Malhotra, and K.Y. Whang, "GRAQULA: A graphical query language for entity-relationship or relational databases," Research Report RC 16877, IBM T.J. Watson Research Center, Yorktown Heights, NY, March 1991.
29. Y. Tonomura and S. Abe, "Content oriented visual interfaces using video icons for visual database systems," Proceedings of the IEEE Workshop on Visual Languages, Roma, Italy, 1989, pp. 68–73.
30. K. Tsuda, M. Hirakawa, M. Tanaka, and T. Ichikawa, "Iconic browser: An iconic retrieval system for object-oriented databases," Journal of Visual Languages and Computing, vol. 1, no. 1, 1990, pp. 59–76.
31. J.D. Ullman, "The U.R. strikes back," Proceedings of the ACM Principles of Database Systems, Los Angeles, California, 1982, pp. 10–22.
32. S. Weissman, "Changing query by icons to improve querying processing for mobile users," M.S. Project, University of Pittsburgh, May 1995.
33. K.Y. Whang, A. Malhotra, G.H. Sockut, L.M. Burns, and K.S. Choi, "Two-dimensional specification of universal quantification in a graphical database query language," Transactions on Software Engineering, vol. 18, no. 3, 1991, pp. 216–224.
34. Z. Zhang and A.O. Mendelzon, "A graphical query language for entity relationship databases," An Entity-Relationship Approach to Software Engineering, C. Davis, S. Jajodia, P. Ann-Beng NG, and R.T. Yeh (Eds.), North Holland, 1983, pp. 441–448.
35. M.M. Zloof, "Query by example," Proceedings of the National Comput. Conference, 1975, pp. 431–438.