On Graph Based Interaction for Semantic Query Languages

Giuseppe Santucci
Università di Roma "La Sapienza "
Dipartimento di Informatica e Sistemistica
Via Salaria 113, I-00198 Roma, Italy
santucci@dis.uniroma1.it

Abstract

In the last years, several proposals have been presented concerning graphical query languages working on diagrammatic representations of semantic data models. Such proposals are mainly based on two different user interaction modalities, i.e., to allow the user to specify either a path or a view on the database schema . In this paper we analyze these two strategies, showing that they are characterised by complementary advantages and disadvantages; then we present a new graphical query language combining the advantages of both the approaches.

1 Introduction

A central problem in the database area concerns the development of tools that allow non expert users to understand and easily extract information from a database. A key issue, in order to build a bridge between the end user and the data, is to represent the database by means of a semantic data model and to define a query language for it; we call this kind of query languages *semantic query languages*. In this way we obtain full independence not only from the physical implementation of the database but also from the logical model. Moreover, in a semantic data model ([17, 21]) the relationships among objects are explicitly represented, and not "embedded" in obscure attribute equivalences, as in the relational model.

Recently, a noticeable effort has been spent in the definition and the experimentation of semantic query languages, ranging on a wide variety of data models and on different query language expressiveness (see, e.g., [14, 5, 4, 10, 2, 16]). All of the available proposals adopt one between two opposite strategies. The first one, that we call *path-based*, is seated on the idea of constructing the query by specifying a path on the schema; the second one, that we call *view-based*, allows for defining a view on the schema.

The two approaches are characterised by complementary advantages and disadvantages; in order to overcome them, we present a query language which adopts a new strategy for user interaction. Roughly speaking our proposal allows for specifying in an integrated fashion both paths and views, capturing the advantages of the two strategies.

Among the available data models, we have chosen the Entity Relationship model (ER); however, our proposal is general enough to be extended to other types of graph-based data models.

The paper is organized as follows. In section 2 we recall the main features of semantic query languages, analyzing the differences between the path-based and the view-based approaches. In section 3 we introduce our hybrid approach, giving some examples. In section 4 the new query language is formally characterized. Concluding remarks point out open research problems in the area.

2 Semantic Query Languages

The growth of the class of database users, including more and more non-expert and casual users, has motivated the development of graphical query languages that are the main component of friendly interfaces for accessing databases. A number of proposals are concerned with the usage of either a semantic or object data model as a means for representing the information content of the database. The availability of graphical representations of such models results in the natural building of systems in which the user interacts visually with a diagram representing the underlying semantic schema. In most of the above cases, the graphical representation of the semantic model is a graph, and the user interaction is defined in terms of a suitable set of graphical primitives whose semantics is expressed by means of a formal, internal, query language (a general analysis on that is in [7]).

Through the selection of a single concept and the manipulation of its characteristics (attributes) the user is provided with the basic selection and projection operators. Usually, during this phase, context sensitive interaction mechanisms free the user from being acquainted with the syntax of the underlying query language, therefore avoiding syntax mistakes. By selecting pairs of adjacent concepts, i.e., exploiting the graphical representation of relationships among classes, the user can simply specify queries involving the join operator, being unaware of the logical implementation of such relationships. Through this second mechanism the expressive power of the graphical query language reaches the one of the conjunctive queries [8]. To specify a set of adjacent concepts can be seen as the basic building block of the user interaction in building a query, and the way in which the user is allowed to do that in the proposals available in the existing proposals can be reduced to two main paradigms of interaction: path-based (see for instance, [14, 13, 15, 19, 20, 2]) or view-based (see for instance, [26, 5, 11, 18, 12, 16, 7]).

To enhance the expressive power of the query language several strategies are available. The most common one is based on the idea of combining two or more queries through the set-oriented operators of union, intersection, and difference, thus reaching the relational completeness. Moreover, several proposals allow for specifying queries involving linear recursion, giving the query language the expressive power of relational algebra augmented by a transitive closure operator.

It is worth noting that, while the selection of adjacent concepts is performed through the direct manipulation [23] of the graph, languages more expressive than conjunctive queries are based on additional non-graphical operators, like icons, special words, etc. (see [6]), therefore not strictly related to the visual operations performed by the user on the graph. Moreover, such languages use conjunctive queries as the basic building blocks for expressing more complex queries. Thus, in the following, we will concentrate on conjunctive queries.

The semantics given to the selection of linked concepts can be easily characterised by restricted logical calculus formulae, in which only the existential quantifier and the usual propositional operators are available. The way in which the existentially quantified variables are given a scope, under the light of the kind of query the user wants to express, has not been investigated in the literature; a study close to the one presented here is in [25], but there the focus was uniquely on universal quantifiers.

The following subsections will describe the implicit assumptions made by the two paradigms of interaction (view-based and path-based) showing how they manage existentially quantified variables and their scope.

2.1 The path-based approach

In the path-based approach, the user specifies a path among the classes and the relationships of the schema, expressing selection and projection conditions. Roughly speaking, it corresponds to an ordered sequence of joins between the pairs <class, relationship> constituting the path, followed by a final selection and projection[1]. The explicit presence of the relationships prevents the user from looking

[1]Because the attribute projection does not affect the issues the paper concentrates on, in the following we will ignore this activity. On the other hand, we consider selection, neglecting the way in which it is specified by the user

for concepts like "foreign keys" and the system can perform the correct the join operations.

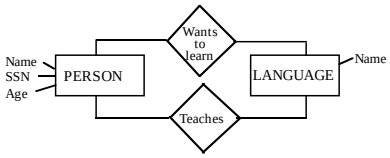

Figure 1: The example of query Q1

The path-based approach is particularly useful when the query needs to involve the same class more than once. In that case, the user (and the system) can distinguish the different occurrences of the class by simply considering its position in the path. For instance, consider the ER schema of Figure 1 and assume the user query is "Q_1: Find out all the possible tuples <student, language, teacher>." The user specifies the path <PERSON, WANTS TO LEARN, LANGUAGE, TEACHES, PERSON>, distinguishing teachers from students looking at the different position of the entity PERSON in the path. More formally:

$Q_1 = \{<p_1, l, p_2> \mid \exists p_1, p_2 \in$ PERSON $\wedge\ \exists l \in$ LANGUAGE $\wedge\ <p_1, l> \in$ WANTS TO LEARN $\wedge\ <p_2, l> \in$ TEACHES$\}$

Note that, according to the structure of the query it is possible to get in the result teachers teaching themselves (in the case in which a teacher wants to learn a language s/he is teaching). To avoid that unlikely situation it is necessary to add the condition $p_1 \neq p_2$.

The above example shows that the path-based approach allows for existentially quantifying an unbounded number of variables for each class, specifying their scope, i.e., the relationship(s) in which the variables must appear. Each time the user path involves a class, say c, a new variable is existentially quantified for that class; such a variable is required to appear in the incoming and in the outcoming relationships of the class c in the path. If the path involves n times a class, as in query Q_1, n different variables are quantified.

Specifying two different variables for the entity PERSON is very useful for expressing query Q_1, but the above approach exhibits its limits when the user has to select a class twice because of the topological structure of the schema, as shown in Figure 2.

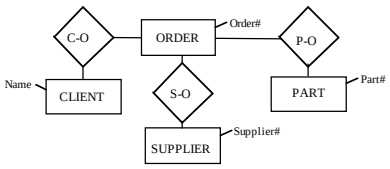

Figure 2: Second example ER schema

If the user wants to know the whole set of <client, part, supplier> belonging to a specified order (say 1021) s/he has to specify the (unnatural) path <CLIENT, C-O, ORDER, S-O, SUPPLIER, S-O, ORDER, P-O, PART>, with the condition that both the first and the second occurrence of ORDER are the same. More formally:

$Q_2 = \{<c, o_1, o_2, s, p> \mid \exists c \in$ CLIENT $\wedge\ \exists o_1, o_2 \in$ ORDER $\wedge\ \exists s \in$ SUPPLIER $\wedge\ \exists p \in$ PART $\wedge\ <c, o_1> \in$ C-O $\wedge\ <s, o_1>, <s, o_2> \in$ S-O $\wedge\ <p, o_2> \in$ P-O $\wedge\ o_1 = o_2 \wedge o_1.Order\# = 1021\}$

Putting the condition $o_1 = o_2$ presents two major drawbacks: (1) it confuses the user that does not suppose s/he is handling two different orders and (2) it requires an additional mechanism for relating different occurrences of the same concept in the path.

The problem is due to the fact that the user touches the entity ORDER twice (thus quantifying two different variables for that class) because this is the only way to reach through a connected path the entity PART from CLIENT involving the entity SUPPLIER as well.

In most of the proposed graphical query languages the user is allowed to specify multiple paths starting from a single concept (see, e.g., [22]) relating them through set-oriented operators. Following this approach, the above query can be expressed stating three paths from the entity ORDER, one reaching CLIENT, one reaching SUPPLIER, and the last one reaching PART. As a matter of fact, however, also this strategy is not natural.

The above example helps us in pointing out the drawbacks which with we are dealing. The problem is not related to the expressiveness of the language

(it is possible to express the query) but it comes out of the unnatural way in which the query has to be expressed.

2.2 The view-based approach

The view-based approach shows complementary advantages and disadvantages with respect to the path-based one. In this case, the user specifies a view and the query corresponds to an unordered sequence of "natural joins". As an example, the query corresponding to the view of Figure 1 coincides with the natural join of the two relationships WANTS TO LEARN, TEACHES on all the shared entities. In this case the two relationships share both PERSON and LANGUAGE, thus the query corresponding to that view furnishes as answer all the pairs <person, language> where the person teaches **and** wants to learn the related language. In other words, we are giving the following semantics to a view: existentially quantify a variable for each entity and assume that the variable is involved in all the adjacent relationships. More formally the semantics of the query associated with the view of Figure 1 is the following:

$Q_3 = \{<p, l> \mid \exists p \in \text{PERSON} \wedge \exists l \in \text{LANGUAGE} \wedge <p, l> \in \text{WANTS TO LEARN} \wedge <p, l> \in \text{TEACHES}\}$

Using the view-based strategy, if a class has to be involved twice or more in the query it is necessary to duplicate the class. Referring to query Q_1, to obtain the desired result, it is necessary to duplicate the entity PERSON, giving rise to the (possibly confusing) view of Figure 3.

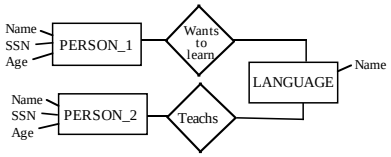

Figure 3: The query Q1 in the view-based approach

The semantics of the query is the following:

$Q'_1 = \{<p_1, l, p_2> \mid \exists p_1 \in \text{PERSON\_1} \wedge \exists p_2 \in \text{PERSON\_2} \wedge \exists l \in \text{LANGUAGE} \wedge <p_1, l> \in \text{WANTS TO LEARN} \wedge <p_2, l> \in \text{TEACHES}\}$

Conversely, to compute the second query of section 2.1, the user must select exactly the view of Figure 2, with the associated semantics:

$Q'_2 = \{<c, o, s, p> \mid \exists c \in \text{CLIENT} \wedge \exists o \in \text{ORDER} \wedge \exists s \in \text{SUPPLIER} \wedge \exists p \in \text{PART} \wedge <c, o> \in \text{C-O} \wedge <s, o> \in \text{S-O} \wedge <p, o> \in \text{P-O} \wedge o.Order\# = 1021\}$

that corresponds to the natural join of the relationships C-O, S-O, and P-O on the shared entity ORDER.

It is worth pointing out that, when the adopted data model supports reflexive relationships, ambiguity in evaluating the query using the view-based approach can easily arise. We will clarify this statement through an example, based on the ER schema of Figure 4.

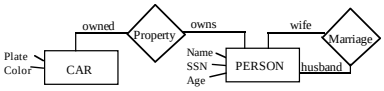

Figure 4: Disambiguating reflexive relationships

Using the path-based approach, the path <CAR, PROPERTY, PERSON, MARRIAGE (through the husband role), PERSON> plus the selection condition PERSON[1].age$\geq$ 25[2] will give all the cars owned by married men older than 25 together with their wives; conversely, the path <CAR, PROPERTY, PERSON, MARRIAGE(through the wife role), PERSON> plus the condition PERSON[1].age$\geq$25 will give all the cars owned by married women older than 25 together with their husbands. Both paths, in fact, cause the existential quantification of two variables for the entity PERSON; the former specifies that the person involved in the relationship property and in the condition age$\geq$25 is the husband, the latter the wife. More formally, the semantics of the first path is:

$Q_3 = \{<c, p_1, p_2> \mid \exists c \in \text{CAR} \wedge \exists p_1, p_2 \in \text{PERSON} \wedge <c, p_1> \in \text{PROPERTY} \wedge <p_1, p_2> \in \text{MARRIAGE} \wedge p_1.age \geq 25\}$

while the semantics of the second path is:

$Q_4 = \{<c, p_1, p_2> \mid \exists c \in \text{CAR} \wedge \exists p_1, p_2 \in \text{PERSON} \wedge <c, p_1> \in \text{PROPERTY} \wedge <p_2, p_1> \in \text{MARRIAGE} \wedge p_1.age \geq 25\}$

[2]We denote with concept[i] the occurrence i of the concept in the path

where we have assumed that the pairs belonging to the relationships MARRIAGE have the form of <husband, wife>.

Let us consider, in the view-based approach, the query corresponding to the view of Figure 4 plus the condition PERSON.age≥25. Because of the presence of the reflexive relationships we have now to quantify two variables for the entity PERSON and the problem is to determine which of them is involved in the PROPERTY relationship and is affected by the PERSON.age≥25 condition. Obviously, there is no way to guess the right answer, and a piece of additional information is needed. We can, for instance, assume that the MARRIAGE relationship is oriented in the sense person-husband-marriage-wife-person, thus giving to the query the meaning associated with the first of the two above paths. Thus, to fully support the view-based approach, we have to assume that there is an implicit order of the occurrences of the same entity in a reflexive relationship and, to the best of our knowledge, no semantic data model supports this kind of information. Note that the relationship roles do not help us to disambiguate the query: they just allow us to distinguish the multiple occurrences of an entity within a relationship and not for determining their scope in the query.

Perhaps the above problems have led to the development of view-based semantic query languages in which reflexive relationships are not allowed (see, e.g.,[26, 5, 7]).

In order to include reflexive relationships we need to slightly modify the semantics of the view-based approach. In particular, we have to consider more than one variable existentially quantified for an entity if it is involved in one or more reflexive relationships. The number of variables depends on the arities of the reflexive relationships involving that entity. More precisely, given an entity e, on which $r_1, \ldots, r_k$ reflexive relationships are defined , with arities $a_1, \ldots, a_k$, we have to consider $1 + \sum_{i=1}^{k}(a_i - 1)$ existentially quantified variables for e. Moreover, we assume that, for each entity involved in a reflexive relationship relationship, has been stated which is the first role. As an example, let us consider the view of Figure 5 in which the arrows denote the roles that must be considered as the first ones in the relationships.

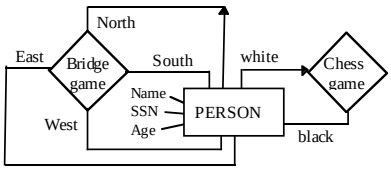

Figure 5: Reflexive relationships on the same entity

Using the proposed semantics, the query will return all the persons playing both bridge games (as North) and chess games (as white) together with the other three bridge players and their black chess antagonists. In fact, we have existentially quantified five variables for the entity person (one, plus three for the bridge game and one for the chess game), binding the two relationships only on the first occurrence, i.e., the North and white player. More formally:

$Q_5 = \{<p_1, p_2, p_3, p_4, p_5> \mid \exists p_1, p_2, p_3, p_4, p_5 \in$ PERSON $\wedge <p_1, p_2, p_3, p_4> \in$ BRIDGE GAME $\wedge <p_1, p_5> \in$ CHESS GAME$\}$

The adopted strategy in evaluating reflexive relationships is only an arbitrary (but very reasonable) choice among a set of more complex ones: as an example, we can assume that **all** the variables existentially quantified for an entity involved by the reflexive relationships must be considered (both in relationships and conditions), giving to view on Figure 4 the meaning of all the cars owned by a person older than 25 married with another person older than 25 that owns a car as well. Note that, in this case, no ambiguity arises during the evaluation of the query. Nevertheless, we strongly believe that to require only one of the variables existentially quantified for an entity involved by reflexive relationships is a member of all the incoming relationships and that it is affected by the selection conditions is a very natural choice and, in the case of multiple reflexive relationships involving the same entity with different arities, it is the only reasonable one. As an example, considering again the view of Figure 5, another (unnatural) approach should be as follows:

1. fully order each reflexive relationship;
2. consider a number of variables corresponding to

the highest arity of the reflexive relationships;

3. for each relationship bind as many variables as the arity of the relationship.

Following this approach and assuming the order <North, South, West, East> for the BRIDGE GAME relationship, the query will result in all the pairs of chess players that also play bridge games together, the white as North and the black as South plus their West an East antagonists. The above example shows that in principle it is possible to bind the relationships on a ranging number of variables giving to the query a totally non intuitive meaning.

In the following we will adopt the semantics we have chosen in evaluating query Q_5; we want to point out, however, that this choice does not affect the following discussion and that the results we will show are valid for any different semantics we give to the view-based approach.

3 Combining views and paths

In this section we introduce our proposal, i.e., an interaction strategy based on a combination of the two strategies discussed in the previous section. We first give an intuitive semantics of the new approach, then we show how the user interaction with a visual interface which is based on it looks like.

3.1 The hybrid approach

The above comparison between the path-based and the view-based approach shows that although they share the same expressive power, none of them is well suited for expressing in a natural way all possible queries. In order to overcome this problem we introduce a new query language strategy, that we call *hybrid approach*. Roughly speaking, it allows the user to specify a view and a set of paths starting from it, having the form $<e_0, r_1, e_1, \ldots, r_k, e_k>$, where e_0 belongs to the view and any other entity in the path may or may not belong to the view. The semantics of the hybrid approach corresponds to specify suitable existential quantified variables according to the semantics chosen for the view-based approach plus additional variables corresponding to the entities appearing in each path (except the first one). The scope of the variables coming from the view is the one adopted in the view-based approach; the scope of the variables coming from the paths are the incoming and outcoming relationships in the path itself. For instance, the query Q_1 can be expressed in the hybrid approach by selecting the view {PERSON, WANTS TO LEARN, LANGUAGE} and specifying the path <LANGUAGES, TEACHES, PERSON>.

As a more complex example, assume that, referring to Figure 2 the user query is "Q_6: Find all the parts belonging to the orders involving one supplier who appears in one of the orders of John Smith." Note that in this query we are dealing twice with the entity ORDER: the first one we look at it to find out all the suppliers of John Smith, the second one to find out all the parts sold by one of the suppliers of John Smith. To express that query using the hybrid approach, the user selects the view { CLIENT, C-O, ORDER, S-O, SUPPLIER} (with the condition Client.Name="John Smith"), thus finding all the John Smith's orders and the related suppliers. Afterwards s/he specifies the path <SUPPLIER, S-O, ORDER, P-O, PART> reaching all the orders of the John Smith's suppliers with the associated parts. Considering the view we have three existentially quantified variables, one for CLIENT, one for ORDER, and one for SUPPLIER. The scope of those variables are the relationships C-O and S-O. Considering the path, we have a variable for ORDER and a second variable for PART, whose scopes are S-O and P-O, respectively. More formally:

$$Q_6 = \{<c, o, s, o_1, p> \mid \exists c \in \text{CLIENT} \wedge \exists o, o_1 \in \text{ORDER} \wedge \exists s \in \text{SUPPLIER} \wedge \exists p \in \text{PART} \wedge <c, o> \in \text{C-O} \wedge <s, o> \in \text{S-O} \wedge <s, o_1> \in \text{S-O} \wedge <o_1, p> \in \text{P-O} \wedge c.name = "John\ Smith"\}$$

The above query gives us the feeling of the strategy underlying the hybrid approach: the user can first focus the attention on the fragment of the schema (view) containing the data answering her/his query and, afterwards, s/he concentrates on data needing a navigation on the schema.

3.2 Visual user interaction

A real system, working with the approach discussed in the previous section is under development. The system is an extension of QBD* [2, 22], initially designed under the path-based paradigm. Using the new system, the user is allowed to construct a view and several paths starting from it. Obviously, it is possible that one of the two query components (view or path) is missing, and in this case the hybrid approach is equivalent to the one corresponding to the unique component of the query.

Here we report the way in which the user interacts with the new version of QBD* to express queries in a hybrid fashion.

The user interacts with ER schemata and s/he is provided with two simple mouse-based operations: (1) the selection of a portion of the screen and (2) the selection of a point of the screen.

The first operation allows for constructing a view: each time the user selects a portion of the screen all the entities belonging to that portion are added to the view. Multiple selections of the same concept are ignored and the system helps the user in building a view without disconnected components.

Through the selection of a point the user specifies a single concept to be included in a path. The system checks that all the paths have an entity belonging to the view previously defined as the initial concept and that the last select concept is linked to the previous one.

If the view has not been defined, the system assumes that it coincides with the first entity in the first path and all the consequent paths must start from that entity.

During the query construction, the user is provided with an intuitive feedback on the variables existentially quantified on the schema, together with their scope. In particular, little colored circles represent the quantified variables for a certain entity and colored lines are associated with each circle, graphically denoting the scope of that variable. In figure 6 the feedback corresponding to the query Q_6 is shown.

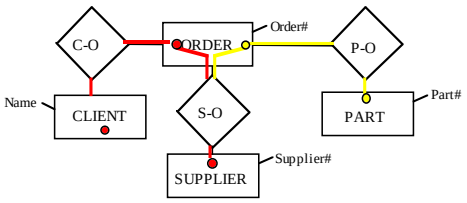

Figure 6: Feedback provided in the hybrid environment

4 The Graphical Query Language

In this section we formally characterise the syntax and the semantics of the proposed hybrid query language. To this end, we first provide a syntax and a semantics for the ER model. Obviously there does not exists a real DBMS directly based on the ER model and a correspondence with a logical model (e.g., relational) must be stated. The problem is not new in the literature, and in [2] a solution particularly suited for the hybrid approach can also be found. So, in the following, we will concentrate exclusively on the ER model, neglecting the underlying logical model.

4.1 Syntax and semantics for the ER model

In what follows we use the term concept to denote an entity, a relationship, or a generalisation hierarchy [3]. We represent an ER schema as a set consisting of:

- entities, each one characterized by a distinct name n belonging to the set EN;
- relationships, with structure R(<name>,<role:name>,<role:name>, [<role:name>]*), where <name> is the name of the relationship belonging to the set RN and <role:name> denotes an involved entity together with its role. We define the function Role(r, e) as that which gives all the existing roles between the relationship r and the entity e and the function FirstRole(r, e) as that which gives the role that must be considered first in

evaluating a reflexive relationship[3]. We define the function Adj(r) as that which gives all the entities adjacent to the relationship r and the function Nadj(r) as that which gives the number of the entities adjacent to r, including repetitions, and with Count(r, e) the number of times the relationships r involves the entity e;

- hierarchies, with structure H(<name>,[<name>]$^+$), where the first name denotes the superset entity, and the others denote the subset entities;
- attributes, with structure ATT(<name>,[<name>]$^+$) where the first name is either an entity or a relationship name and the other ones represent the set of attributes associated with it.

The set oriented semantics of an ER schema is defined as follows.

Let D be a universe of printable, atomic objects, representing the values of the attributes; let O be a universe of unprintable, atomic objects, representing object identifiers; let L be a set of labels useful to denote tuple components; and let R be a set of labels useful to denote relationship roles. Let $FE_1, FE_2, \ldots, FE_n$ be sets of functions between $(L \times O)^i$ and D, characterized by an increasing arity of the function domain. More precisely, a function f belonging to FE_i maps elements of $(L \times O)^i$ into elements of D : $f : (L \times O)^i \rightarrow D$ and we say that f is of rank i. Let $FR_1, FR_2, \ldots, FR_n$ be sets of functions between $(L \times R \times O)^i$ and D, characterized by an increasing arity of the function domain. More precisely, a function f belonging to FR_i maps elements of $(L \times R \times O)^i$ into elements of D: $f : (L \times R \times O)^i \rightarrow D$ and we say that f is of rank i.

The interpretation of an entity is denoted with m(name), and is a set of tuples of the form $<l_1 : o_1,\ l_2 : o_2, \ldots, l_k : o_k>$, where $l_i \in L$ and $o_i \in O$ for i=1...k. The interpretation of a relationship is denoted with m(name), and is a set of tuples of the form $<l_1 : r_1 : o_1,\ l_2 : r_2 : o_2, \ldots, l_k : r_k : o_k>$, where $l_i \in L, r_i \in R$ *and* $o_i \in O$ for i=1...k and we say that k is the rank of the concept. An attribute of an entity of rank k is a function belonging to the set FE_k. An attribute of a relationship of rank k is a function belonging to the set FR_k.

[3]If the function is applied to a non reflexive relationship it returns to the unique role existing between the relationship and the entity

4.2 ER Query

The aim of this section is to define the syntax and the semantics of a query on an ER schema s (ERQ(s) in what follows). The definition of an ERQ(s) is based on the hybrid approach introduced in section 3.1. An ERQ(s) is defined as follows:

ERQ(s) ::=<*atomic query*> [<*entity_name*> {, <*set operator*>, <*atomic query*>}*]
set operator::= $<\cup | \cap | ->$

An atomic ER query is defined as a view on the schema plus a set of paths starting from it:

<*atomic query*>::=<*view(s)*> [, <*path(view(s))*>]*
<*view(s)*>::=<*c_entity_name*>$^+$<*c_rel_name*>$^+$
<*path(view(s))*>::=<*c_entity_name*> [, <*role*>, <*c_rel_name*>, <*role*>, <*c_entity_name*>]$^+$
<*c_entity_name*>::=<*entity_name*> [, <*p*>][, <*f*>]
<*c_rel_name*>::=<*rel_name*> [, <*p*>][, <*f*>]
<*entity_name*>::= an entity
<*rel_name*>::= a relationship
<*p*>::= a set of attributes
<*f*>::= a logical formula

where view(s) is a syntactically correct subschema of s in which each concept is associated with the optional parts [,p] [,f], where p is a subset of the attributes of c and f is a logical formula based on the attributes of c. Moreover, each path(view(s)) starts from an entity belonging to view(s).

In order to define the semantics of an atomic query we need some preliminary definitions.

Let InvoV (involving through view) be a function from the set of entity names EN and an atomic_query to the powerset of the relation names 2^{RN} defined as follows:

$InvoV(e, aq) = \{r | r \in RN \ \wedge view(s) \ \in aq \ \wedge e \in view(s) \ \wedge count(r, e) > \ 0 \wedge r \in view(s)\}$

In words, relationship r belongs to InvoV(e,aq) if it belongs to view(s) and involves e.

Let Invo be a function from the set of entity names EN and an atomic_query to the powerset of the relation names 2^{RN} defined as follows:

$InvoV(e, aq) = Invo(e, aq) \cup \{r | r \in RN \wedge view(s) \in aq \wedge e \in view(s) \wedge count(r, e) > 0 \wedge \exists path(view(s)) = <e, role_1, r, \ldots>)\}$

Therefore, a relationships r involving e belongs to Invo(e,aq) if it belongs to InvoV(e,aq) or if it is the starting relationship of one of the paths associated with view(s) starting from e.

Now we can define the semantics of an atomic query.

1. For each entity $e \in view(s)$, existentially quantify k variables $v_1, vr_{1,2}, \ldots, vr_{1,count(r_1,e)}, \ldots, vr_{n,2}, \ldots, vr_{n,count(r_n,e)}$ where $k = 1 + \sum_{i=1}^{n}(count(r_i, e) - 1)$ and $\{r_1, \ldots, r_n\} = InvoVR(e, aq)$. The scope of v_1 is $FirstRole(r_i, e)$ with $r_i \in Invo(e, aq)$, plus the logical formula f defined on the attributes of e and the logical formulas stated on the attributes of all the relationships belonging to InvoV(e,aq); the scope of $vr_{i,2}, \ldots, vr_{i,count(r_i,e)}$ is the set $Role(r_i, e) - FirstRole(r_i, e)$ of the reflexive relationship r_i belonging to InvoV(e,aq) plus the logical formula stated on the attributes of r_i.

2. For each path(view(s)):

   (a) for each entity in the path e_i (i>1) $<\ldots, r_{i-1}, role_{i-1}, e_i, \ldots>$ existentially quantify a variable ; the scope of this new variable are $role_{i-1}, role_i$, plus the logical formulas associated with the attributes of e_i, r_{i-1}, and r_i;

   (b) for each relationship r_i having $Nadj(r_i) > 2$:

      i. for each $e \in Adj(r_i) - \{e_i, e_{i+1}\}$ existentially quantify $Count(r_i, e)$ variables whose scope is $Role(r_i, e)$;

      ii. existentially quantify $Count(r_i, e_i) - 1$ variables whose scope is $Role(r_i, e_i) - role_i$;

      iii. existentially quantify $Count(r_i, e_{i+1}) - 1$ variables whose scope is $Role(r_i, e_{i+1}) - role_{i+1}$.

3. The above two steps will produce a set of tuples of instances of all the involved entities. The final result is built by applying to each instance the functions corresponding to the attributes specified in the corresponding <p> part.

Step 1 provides for the quantification of the variables for the entities belonging to the view; the scope of those variables is the set of the roles of the relationships belonging to the view plus the roles in first position of all the paths starting from the view.

Step 2(a) provides for the quantification of the variables for the entities belonging to the paths (but the first one); the scope of those variables are the incoming and the outcoming roles in the path.

Step 2(b) is a refinement of step 2(a): because of the path can cross relationships with arity greater than two a path can produce "dangling roles". Step i looks for dangling roles involving entities not belonging to the path; step ii and iii look for dangling roles involving in a reflexive way entities in the path.

5 Conclusion and further research

In this paper we have analyzed the two main strategies adopted by the new generation of semantic query languages, namely the path-based and the view-based approaches. The comparison showed us that neither of them is suitable for handling in a natural way all the useful types of existential quantifications needed for expressing conjunctive queries. As an attempt to overcome the drawbacks of the two approaches we formally defined a new strategy of querying, namely the hybrid approach, combining the advantages of both.

The proposal presented in this paper for the ER model can be easily extended to any graphical query language supporting the notion of class and relationship, enhancing the friendliness of the way in which

the query is expressed and, in some case, also the expressive power of the language itself. As an example, it has been reported by several authors [9, 24] the lack of expressive power of the object oriented query language presented in [18] based on the view-based approach; the inclusion in that environment of the hybrid strategy discussed in this paper could easily solve the problem.

Finally, considering the implementation of a system based on the hybrid approach, we are actually modifying the QBD* environment [2, 22], based on the path-based strategy, in order to extend it to the hybrid approach allowing to test, through usability experiments, the effectiveness of this new strategy.

5.1 Acknowledgements

The author wishes to thank Maurizio Lenzerini and Elisa Janz for their comments on this work.

References

[1] *Proc. 4th Intl. Conference on the Entity Relationship Approach*, Chicago, Illinois, 1985.

[2] M. Angelaccio, T. Catarci, and G. Santucci. Qbd*: a graphical query language with recursion. *IEEE Transactions on Software Engineering*, 16(10):1150+, 1990.

[3] C. Batini, S. Ceri, and S. B. Navathe. *Conceptual Database Design: an Entity Relationship Approach*. Benjamin & Cummings Publishing Company, 1991.

[4] D. Bryce and R. Hul. Snap: A graphics-based schema manager. In *Proc. IEEE Intl. Conference on Data Engineering*, pages 151–164, Los Angeles, USA, 1986.

[5] D. M. Campbell, D. W. Embley, and B. Czejdo. A relationally complete query language for an entity-relationship model. In *Proc. 4th Intl. Conference on the Entity Relationship Approach* [1], pages 90–97.

[6] T. Catarci. On the expressive power of graphical query languages. In *Proc. 2nd IFIP W.G. 2.6 Working Conference on Visual Databases*, pages 411–421. North-Holland, 1991.

[7] T. Catarci, G. Santucci, and M. Angelaccio. Fundamental Graphical Primitives for Visual Query Languages. *Information Systems*, 18(2), 1993.

[8] A. K. Chandra. Theory of database queries. In *Proc. Symp. Principles of Database Systems*, 1988.

[9] I. F. Cruz. Declarative Query Languages for Object-Oriented Databases. In F. H. Lochovsky, editor, *Technical Report CSRI-238*, pages 92–130. CSRI, University of Toronto, June 1990.

[10] I. F. Cruz, A. O. Mendelzon, and P. T. Wood. A graphical query language supporting recursion. In *Proc. ACM-SIGMOD Conference on the Management of Data*, pages 151–164, 1987.

[11] I. F. Cruz, A. O. Mendelzon, and P. T. Wood. G+: Recursive queries without recursion. In *Proc. 2nd Intl. Conference on Expert Database Systems*, pages 355–368, 1988.

[12] B. Czejdo, R. Elmasri, D. Embley, and M. Rusinkiewicz. A graphical data manipulation language for an extended entity-relationship approach. *IEEE Computer*, 23(3), 1990.

[13] R. Elmasri and J. A. Larson. A graphical query facility for er databases. In *Proc. 4th Intl. Conference on the Entity Relationship Approach* [1].

[14] R. Elmasri and G. Wiederhold. Gordas : a formal high-level query language for the entity-relationship model. In *Proc. 2nd Intl. Conference on the Entity Relationship Approach*, pages 49–72, Washington, D.C., 1981.

[15] K. J. Goldman, S. A. Goldman, P. C. Kanellakis, and S. B. Zdonik. Isis: Interface for a semantic information system. In *Proc. ACM-SIGMOD Conference on the Management of Data*, pages 328–342, 1985.

[16] M. Gyssens, J. Paredaens, and D. Van Gucht. A graph-oriented object model for database end-user interfaces. In *Proc. ACM-SIGMOD Conference on the Management of Data*, pages 24–33, Atlantic City, USA, 1990.

[17] R. Hull and R. King. Semantic database modeling: Survey, applications and research issues. *ACM Computing Surveys*, 19(3):201+, 1987.

[18] K. C. Kim, W. Kim, and A. Dale. Cyclic Query Processing in Object-Oriented Databases. In *Proc. IEEE Intl. Conference on Data Engineering*, pages 564–571, 1989.

[19] R. King and S. Melville. Ski: A semantic knowledgeable interface. In *Proc. 10th Intl. Conference on Very Large Databases*, pages 30–37, 1984.

[20] M. Kuntz and R. Melchert. Pasta-3's graphical query language: Direct manipulation, cooperative queries, full expressive power. In *Proc. 15th Int. Conference on Very Large Data Bases*, Amsterdam, Holland, 1989.

[21] J. Peckham and F. Maryanski. Semantic data models. *ACM Computing Surveys*, 20(3):153+, 1988.

[22] G. Santucci and P. A. Sottile. Query By Diagram: a Visual Environment for Querying Databases. *Software Practice and Experience*, 23(3), 1993.

[23] B. Shneiderman. Direct manipulation: A step beyond programming languages. *IEEE Computer*, 16:57+, 1983.

[24] K. Vadaparty, Y. A. Aslandogan, and G. Ozsoyoglu. Towards a Unified Visual Database Access. In *Proc. ACM-SIGMOD Conference on the Management of Data*, pages 357–366, 1993.

[25] K. Y. Whang, A. Malhotra, G. H. Sockut, L. Burns, and K. S. Choi. Two-dimensional specification of universal quantification in a graphical database query language. *IEEE Transactions on Software Engineering*, 18(3), 1988.

[26] H.K.T. Wong and I. Kuo. Guide: Graphical user interface for database exploration. In *Proc. 8th Intl. Conference on Very Large Databases*, pages 22–32, 1982.