The visual query language CQL for transitive and relational computation

Kalervo Järvelin a,*,1, Timo Niemi b,2, Airi Salminen c,3

a Department of Information Studies, University of Tampere, P.O. Box 607, FIN-33101 Tampere, Finland
b Department of Computer Science, University of Tampere, P.O. Box 607, FIN-33101 Tampere, Finland
c Department of Computer Science and Information Systems, University of Jyväskylä, P.O. Box 35, FIN-40351 Jyväskylä, Finland

Received 8 April 1998; received in revised form 2 February 1999; accepted 3 March 2000

Abstract

Classification query language (CQL) is a high-level visual query language with a great expressive power. In CQL the processing of ordinary relations and classifications based on transitive relationships is integrated seamlessly. Relations and classifications are represented in the visual interface in a uniform way through relation and classification skeletons. All query formulation in CQL is QBE-like – based on the intuitive way of filling constants and sample values into the skeletons. In order to guarantee great expressive power, relational and classification expressions can be nested freely with each other at unlimited nesting levels. Recursive definition of transitive processing is totally hidden from the users. The query interface and its implementation are described briefly. © 2000 Elsevier Science B.V. All rights reserved.

Keywords: High-level query languages; Visual query languages; Deductive databases; Relational databases; Query-By-Example

1. Introduction

Traditional DBMSs cannot produce all information the users need, although this information is derivable from the stored data. One solution to increasing the expressive power of DBMSs is to allow recursion both in data definition and queries. Logic has been shown an excellent way to define all stored and derived information in deductive databases. Although logical rules have been proposed as a uniform and declarative language for formulating recursive and non-recursive queries [11], recursive logic-based query formulation is too demanding for an ordinary database user. This means that we must develop a more convenient query formulation way for users.

A common approach to solve the problems of logic-based query formulation is to provide the computation of the transitive relationships and embed it into textual or visual query constructs.

*Corresponding author. Tel.: +358-31-21-56953; fax: +358-31-21-56560.
E-mail addresses: kalervo.jarvelin@uta.fi (K. Järvelin), tn@uta.fi (T. Niemi), airi@cs.jyu.fi (A. Salminen).
1 Web: http://www.uta.fi/~likaja/index.html
2 Web: http://www.cs.uta.fi/~tn/index.html
3 Web: http://www.cs.jyu.fi/~airi/

0169-023X/00/$ - see front matter © 2000 Elsevier Science B.V. All rights reserved.
PH: S0169-023X(00)00021-5
Many authors propose that the transitive closure be incorporated as a new operation into the expressive power of conventional query languages, e.g., into QBE [20], SQL [3] and QUEL [6]. There are several visual languages supporting first-order queries, i.e., languages with the expressive power of relational algebra. Catarci et al. [2] classify the visual paradigms of such languages as form-based, diagrammatic, iconic and hybrid. However, there are only a few visual query languages which allow the user to perform non-first-order queries, e.g., Query by Diagram* (QBD*, [1,16]) and G+ [4,12]. They are based on first-order logic extended by a transitive closure operation. In principle, the visual query languages Pasta-3 [10] and visual query language (VQL) [19] provide a greater expressive power. However, in order to specify complex recursive queries in Pasta-3 the user has to formulate Prolog rules in a separate window or, alternatively, simulate quantifiers with textual icons. This is a heterogeneous approach to query formulation and does not, from the user viewpoint, solve the difficulties related to logic-based query formulation. The idea of VQL is to offer one generic visual query language for different data models (e.g., relational, nested, object-oriented, etc.) and for a wide class of recursive queries. VQL provides a uniform approach to different types of queries but it lacks implementation.

Hierarchical classifications constitute a commonplace application area for recursive definition. Classifications are used in many database environments, e.g., in databases for science and technology (e.g., properties of chemical compounds), business and trade (e.g., data on marketing, banking or shares), and social science and statistics. For example, in statistical databases, data on product types, chemical compounds, labor or geographic areas may be classified multi-dimensionally through several classifications. In this paper, we show how classifications can be incorporated in a visual interface, introduce a high-level visual query language, called classification query language (CQL), and briefly describe its implementation. The language integrates both relational and transitive expressive power seamlessly into a single framework. Below we list the features of CQL:

- declarative query formulation,
- great expressive power and clear semantics through high-level visual constructs,
- encapsulation of recursion in concepts which are natural to interpret in the application domain of interest,
- a single homogeneous and uniform QBE-like way of query formulation,
- clear surfaces for implementation, i.e., it offers easy implementation in homogeneous and heterogeneous database environments.

We chose a QBE-like interface style, because querying by example is widely accepted despite some limitations [14]. In the terminology by Catarci et al. [2] our visual approach may be called form-based.

The next section presents our sample database environment. The visual interface is described in Section 3 and in detail in [15]. Section 4 introduces the underlying textual query language used in the implementation (described in detail in [7]). The translation process from the visual to operation-oriented textual queries is outlined in Section 5 (described in detail in [15]). Section 6 contains discussion and conclusions.
2. Data organization

Our classified sample database provides data on air pollution in Great Britain. The data are received from sensor stations at various locations in Great Britain. Each town or city with its surroundings is called a location. In the sample database hierarchic classifications are used to categorize some of the data stored in relational tables. The basic data consist of tables where some of the columns contain values appearing in one or more classifications.

A hierarchic classification contains immediate and indirect subclass and superclass relationships among its elements, which are names called classes taken from a common domain. Fig. 1 shows a tree visualization of a classification for air pollutants. The names in the tree are the classes of the classification. The classification consists of four hierarchic levels and the levels which are expressed explicitly by level names. When we count levels of a classification we consider the root level as level 1. In the classification of Fig. 1, the levels 1–4 are called Air pollutants, Type, Family and Pollutant.

The classifications 'Air pollutant and 'Health hazards (Fig. 2) are for chemical compounds. In both classifications, the lowest level is called 'Pollutant indicating that they have common classes. 'Air pollutants classifies all pollutants and 'Health hazards those pollutants which are considered hazardous. Another sample classification is geographic and classifies
sensor stations by locations, subregions and regions (Fig. 3). The binary relations are indicated in Figs. 2 and 3.

A whole classification in CQL is named by its top-level and characterized by its level names. Each classification is represented as a collection of binary relations. The classification ‘Air pollutants’ is represented by three binary relations pollutant_types (on ‘Air pollutants × Type), pollutant_families (on Type × Family) and pollutants (on Family × Pollutant).

A classification can also be described as a set of paths. A classification path is a sequence \( \langle c_1, \ldots, c_k \rangle \) of classes starting from the root and ending to a leaf class and where each class \( c_i, i = 1, \ldots, k - 1 \), is an immediate superclass of \( c_{i+1} \). For example, in the classification of Fig. 1, ‘Air pollutants,’solid pollutant,dust,ash) is a path, i.e., it contains those classes which are needed to connect the root and a leaf class to each other. Thus it expresses transitive relationships among classes.

In many classification operations, the scope of class processing matters. A scope is a collection of binary relations which represent consecutive classification levels. For example, \{pollutants, pollutant_families\} is a scope.

In our sample database, the tables ‘Critical content’ and ‘Measurements’ are intended for conventional relational processing (Fig. 4). The table ‘Measurements’ shows the average pollutant air content data for pollutants in specific sensor stations and dates. The table ‘Critical content contains data on the critical concentration (Limit) of various pollutants for various recipients. The column headings ‘Pollutant’ and ‘Station’ appear as level headings.

![Sample hierarchy](image)

Fig. 3. A partial geographical classification of sensor stations.

<table>
<thead>
<tr>
<th>Measurements</th>
<th>Critical content</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Pollutant</strong></td>
<td><strong>Measurement</strong></td>
</tr>
<tr>
<td>SO2</td>
<td>0.0000005</td>
</tr>
<tr>
<td>SO2</td>
<td>0.0000001</td>
</tr>
<tr>
<td>NO2</td>
<td>0.0000007</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
</tr>
</tbody>
</table>

Fig. 4. Tables in the sample database.
in classifications as well indicating that the column values are classified. Note that all data in our approach is represented relationally.

3. The visual CQL interface

CQL queries are either (pure/integrated) table queries or (pure/integrated) classification queries. A table query produces one or more tables, and a classification query one or more classifications (or class sets). An integrated query operates on tables and classifications, producing either tables or classifications. Thus in a table query, classifications may be used for constraining the tuples to be retrieved.

Fig. 5 presents Sample Query 1 which is an integrated classification query on types of serious health hazards that have been caused by gas pollutants in the London area in 1993. A health hazard caused by a pollutant is considered serious if its concentration has recently exceeded the critical limit for humans at least once.

![Sample Query 1]
In the query, three classification skeletons are displayed in a separate dialog window called Classification Query. The first item under each classification name indicates classification levels. By pressing the labeled button, values for the level can be viewed and selected. The second item is used for constant limiters and the third for sample value limiters. The last item is a tick box for specifying printing. There are no comparison operators in classification skeletons (equality is assumed).

Sample Query 1 specifies a subclassification of the classification ‘Health hazards, i.e., a classification limited by the sample value ‘Pol associated with the classification level Pollutant must be constructed. The output consists of the classification heading ‘Health hazards and the ‘Hazard type (ticked for printing) which are the superclasses of the classes to which the sample value ‘Pol refers. The result is displayed in ‘Result Window in Fig. 5 and indicates that the hazardous gas pollutants in London are poisons and allergens.

Tables are used in CQL in a way similar to QBE, i.e., using table skeletons [20], whose layout is however different. In a classification query they are available upon request, by clicking the Tables button. A table skeleton shows the name of a table, the column headings of the table, and for each column three items which the user may fill. The first among them is intended for an operator; the default is ‘=’. Other binary operators defined for the domain of the attribute may also be used, e.g., <, >, >=, <=, and ≠. The second item is for a constant value and the third item for a sample value.

The sample limiter ‘Pol appears in two classifications in Sample Query 1. Because the classification ‘Health hazards is ticked for output, the classification ‘Air pollutants is used to constrain it. In the latter, the sample limiter ‘Pol obtains pollutants of the type gas_pollutants as its values because of the constant limiter ‘pollutants. Moreover, because ‘Pol appears also as a sample value in the relation skeletons, the relations must be joined on the ‘Pollutant attributes. The pollutant names for ‘Pol obtained through the relations must be intersected with those gained through the classification ‘Air pollutants. More sample value sharing occurs among the relations. Sharing the sample value ‘Critical between the relation skeletons ensures identifying pollutants whose measurements have exceeded the critical content for ‘humans (a constant limiter for the attribute ‘Recipient). Sharing the sample value ‘Date between the ‘Measurements skeleton and the condition edit window (beneath the relation skeletons) enforces dates within 1993. Sharing the sample value ‘Stat between the ‘Measurements skeleton and the ‘Great Britain classification skeleton enforces sensor stations in London to be considered.

4. The textual operations

Query languages for graphs typically have the capability of querying paths. Often these languages are processed so that queries are translated into some operation-based language (see, e.g., [17] in the case of the object-oriented data model). We apply this principle in the implementation of the CQL. It means that a visual CQL expression is translated into a textual expression consisting of relational algebra (RA) operations and classification operations defined in [13]. Our classification operations can be grouped into the following groups.

The non-directed non-transitive operations contain, e.g., the set intersection operation:
• \textsc{set.intersection}(\text{Class-set1}, \text{Class-set2}) which yields the intersection of the sets \text{Class-set1}, \text{Class-set2}.

The transitive operations contain, e.g., superclass-directed operations, e.g.,

• \textsc{union.of.superclasses}(\text{Class-set}, \text{Scope}) computes all superclasses of \text{Class-set} within \text{Scope}.

  subclass-directed operations, e.g.,

• \textsc{subclasses}(\text{Class}, \text{Scope}) gives all subclasses of a class within the \text{Scope}.

• \textsc{intersection.of.subclasses}(\text{Class-set}, \text{Scope}) computes common subclasses of \text{Class-set} within \text{Scope}.

  path-oriented operations, e.g.,

• \textsc{paths.through.class.set}(\text{RootBinRel}, \text{LeafBinRel}, \text{Class-set}, \text{Scope}) gives, within \text{Scope}, all classification paths constructable between the superclass domain of \text{RootBinRel} and the subclass domain of \text{LeafBinRel} so that each path contains at least one class of \text{Class-set}.

The RA operations consist of the conventional operation set. In Sample Query 1, the operations \textsc{product}, \textsc{join}, \textsc{restriction} and \textsc{projection} are used. Their syntax is assumed self-explanatory.

5. Query translation

CQL query translation is based on a two-phase template-driven translation technique. In the first phase, the form-based visual user query is translated into a set of templates, which are textual equivalents of the visual query components. We have developed a general template structure which is able to represent all visual information textually and uniquely, e.g., interaction of sample limiters and sample values, for query construction concisely. In the second phase, the template structure is used to derive, through a recursively defined process, a nested expression consisting of the operations introduced above. This recursive process allows the construction of nested expression without limiting the number of nesting levels. Thus all nested and transitive query definition, given intuitively and implicitly in the visual query, can be translated in a general way into a, possibly complex, operation-oriented expression [15].

We shall illustrate the query translation process by Sample Query 1. The whole construction process of the textual expressions for Sample Query 1 is visualized in Fig. 6. The thin arrows represent function calls leaving shaded expression slots for which an expression is needed. The bold arrows bring the constructed subexpressions to the slots.

Sample Query 1 is a path query, i.e., it produces a set of classification paths to be visualized as a hierarchy. It is interpreted as a path-oriented query because the user has specified (Fig. 5) output from the health hazard classification and ticked the print box for ‘Hazard type’. This tick causes also that the sample limiter ‘qvarl’ is associated automatically with the ‘Hazard type’ level indicating that the result paths must contain a value for ‘qvarl’ at this level. Therefore, query construction starts as a path-oriented expression with the path limiter ‘qvarl’ and classification level range from ‘Health hazards’ to ‘Hazard type’ within the classification ‘Health hazards’. Because result paths must contain a value for the path limiter ‘qvarl’ the root operation becomes \textsc{paths.through.class.set}. 

The expression for the path limiter 'qvar1' must be constructed as a class-oriented expression because 'qvar1' is associated with a classification. The class-oriented expression consists of the operations UNION_OF_SUPERCLASSES and SET_INTERSECTION. The former is needed because 'qvar1' is associated with hazard types and its subclass level Pollutant has an associated sample value 'Pol'. Because 'Pol' also occurs both in a class and a relation skeleton, two subexpressions are needed and their results must be intersected. The intersection will represent subclasses of gas pollutants and pollutants which have exceeded a critical value for humans. The first subexpression is a subclass-oriented expression SUBCLASSES('gas pollutant, {pollutant_families, pollutants'}). The other is a relational expression constructed for the sample value 'Pol'. In a normal form, it has the structure PROJECTION(RESTRICTION(PRODUCT('Critical content, Measurements), QBECond), {M.Pollutant}). (The initial letters C and M of the relations are used here and below to prefix the attribute names for uniqueness.)
In forming the predicate QBECond, the join condition (C.Pollutant = M.Pollutant) is constructed, because the two relation skeletons share the sample value 'Pol. The second (non-equi) join condition (Limit < Measurement) is based on sharing the sample value 'Critical between the skeletons and the comparison operator ' <. The conditions (M.Date > 930100) and (M.Date < 940100) are constructed because the sample value 'Date in the skeleton for Measurements also occurs in the condition specification. The condition (Recipient = humans) is based on the constant 'humans in the skeleton for 'Critical content. Finally, a condition is constructed for the attribute 'M.Station. The sample value 'Stat occurs as the value for Station in a class template, which has a limiter 'london for Location. Thus the class-oriented expression SUBCLASSES(london, {gb_stat_names}) is constructed for 'Stat and the condition is thus (M.Station one_of (SUBCLASSES(london, {gb_stat_names}))) specifying any London station.

After algebraic optimization [18] for moving restrictions and projections, the textual expression for visual Sample Query 1 of Fig. 5 has the form given in Fig. 7. It is ready for execution as described in [7]. This sample query illustrates the deductive power of the CQL language in giving as result a higher-order answer than the plain relational table contents. The general processing of all types of CQL queries is defined in [15].

```
PATHS_THROUGH_CLASS_SET(hh_types, hh_types,
UNION_OF_SUPERCLASSES(
  SET_INTERSECTION(
    PROJECTION(
      JOIN(
        PROJECTION(
          RESTRICTION(Critical content, (Recipient = humans)),
          {C.Pollutant, Limit}),
        PROJECTION(
          RESTRICTION(Measurements,
            (M.Date > 930100) and (M.Date < 940100) and
            (M.Station one_of(SUBCLASSES(london, {gb_stat_names})))),
          {M.Pollutant, Measurement}),
        (Limit < Measurement) and (C.Pollutant = M.Pollutant)),
          {M.Pollutant}),
        SUBCLASSES('gas pollutant', {pollutant_families, pollutants}),
          {haz_compounds}),
          {hh_types})))
```

Fig. 7. The operation-oriented nested expression translated for Sample Query 1.
6. Discussion and conclusion

The sample query construction process exemplifies the principles of translation from visual constructs of CQL on a high abstraction level into textual expressions of our operation-oriented query language intended for advanced database applications. Thereby it shows how hierarchic classifications (and transitive relationships in general) can be integrated conveniently in high-level interfaces. This is important in many application domains. Next we reconsider how CQL satisfies the highly desirable properties of a visual query language.

6.1. Visual abstraction level

In CQL, the basic visual constructs are skeletons for tables and classifications and represent ordinary relations and hierarchical classifications. Both constructs can be interpreted in a natural way in many application domains and thus they support the user’s intuition. With CQL, transitive computation is embedded into the classification skeleton construct.

The VQL [19] visual query language has a great expressive power. However, the abstraction level of the basic visual constructs is that of Datalog clauses which seem too low for ordinary database users and lead easily to great complexity in query formulation.

Catarci et al. [2] consider multi-paradigmatic – form-based, diagrammatic, iconic and hybrid – visual access to databases. This would avoid visual complexity because the user may choose the most suitable visualization. However, Catarci et al. explicitly consider only conjunctive first-order queries. The lacking transitive computation is the challenging part included in CQL without visual complexity. In CQL all information for conventional or classification processing is visualized in a tabular form. This high-level representation provides visual constructs that are compatible with each other.

Fig. 7 shows that Sample Query 1 corresponds to a large nested expression. However, even the operations in this expression are at a much higher level than logic-based rules. Therefore, query formulation in CQL is at an essentially higher abstraction level than in deductive database systems. To express Sample Query 1 in the latter would require a large nested logical expression containing several recursive definitions.

6.2. Encapsulating recursion

The skills required from the users are one of the main issues in visual query languages allowing recursion. In this respect the visual query languages differ considerably. For example,

- in G+ [12] query formulation is based on graphical pattern matching and user-specified regular expressions;
- in QBDS the formulation of recursive queries requires the user to navigate in the E-R schema, select the entity of interest, specify the beginning of recursive session, specify the cycle conditions on entity attributes and give conditions to identify a subset of the result of a transitive closure [1].

It is obvious that regular expressions or the specification of recursion are too demanding for ordinary database users. In CQL, the definition of recursion is naturally embedded into basic concepts of classifications and set theory. The users only need to understand database application specific classification concepts and they need not define recursion at all in their queries.
6.3. **QBE-like query formulation**

CQL provides relation and classification skeletons and allows sample value-based integrated queries in the QBE-style. Because all query formulation in CQL is QDE-like, CQL provides homogeneous query formulation also in queries where conventional and classification processing are integrated. Users need to master only the semantics of classifications and table columns. This essentially simplifies query formulation both with respect to textual query formulation (cf. Sample Query 1) and query systems based on RA operations augmented by one generic transitive closure operation [5]. It also avoids complex visual structures with which the user may need to acquaint herself in $G^+$ and QBD*.

6.4. **Implementation**

The CQL implementation is based on the deductive paradigm. Deductive database systems can be implemented on the basis of the homogeneous or heterogeneous approach. Our implementation approach supports both. CQL was implemented in the homogeneous approach, i.e., the deductive and data retrieval parts were both implemented in Prolog. For a heterogeneous implementation, the surfaces between the deductive part (Prolog) and data retrieval part (an existing relational database system) have been defined precisely in [7].

6.5. **Expressive power**

The visual CQL interface provides the expressive power of linear recursive rules which has been considered sufficient in several deductive databases. This expressive power is considerably greater than the pure relational expressive power because it allows the processing of transitive relationships among data. Järvelin and coworkers [8,9] have shown how unlimited but directed transitive computation can be integrated with non-first-normal-form (NF2) processing within a high-level declarative textual query language. A similar QBE-like interface can be provided also in this case.

6.6. **Conclusion**

The visual query language CQL incorporates the processing of ordinary relations and hierarchical classifications within a single framework. To guarantee a great expressive power, many-sided transitive computations are allowed among classes and relations. One homogeneous QBE-like way of visual query formulation is used. This approach integrates two kinds of computations seamlessly with each other without limiting nested computations. Recursive definition typical of transitive processing is totally hidden from the users through high-level visual constructs of CQL. CQL embeds all transitive computation into classifications.

**References**


Kalervo Kal Järvelin is a Professor and Head of the Department at the Department of Information Studies, University of Tampere, in Finland. He received his Ph.D. in Information Studies from the University of Tampere in 1987. Since 1987 he has worked as Associate Professor, and since 1996 as Professor in the Department of Information Studies, University of Tampere. Kal is the leader of the information retrieval research group FIRE (see http://www.info.uta.fi/re-search/fire.html). His current research interests include information retrieval (especially linguistic and conceptual methods in information retrieval), structured documents, user interfaces, and high-level query languages and interfaces. He has been the supervisor of six completed Ph.D. projects in 1995–2000. Kal has published extensively on information retrieval and database management, especially structured objects and their restructuring; high-level data models integrating relational, hierarchical and deductive processing; linguistic and conceptual methods in information retrieval; interfaces for information retrieval; and information needs and seeking.

Airi Salminen is a Professor at the University of Jyväskylä, Department of Computer Science and Information Systems, in Finland. She received her Ph.D. in Computer Science from the University of Tampere in 1989. During 1990 and 1991 she worked as a visiting research associate at the University of Western Ontario in Canada. She was responsible for planning a new Master’s program in Digital Media at the University of Jyväskylä and has headed the program from its beginning in 1995. The main goal of the program is to educate experts for the management and effective use of digital information in networked organizations. She has been the leader of several collaborative projects where the research has been tied to the document management development efforts in major Finnish companies or public sector organizations. Her current research interests include document management in organizations, structured documents, user interfaces, and software environments.
Timo Niemi received his Ph.D. degree from the University of Tampere in 1985. He has worked as Assistant Professor in the Department of Computer and Information Sciences of the same university since 1986. His research interests cover databases (including different database paradigms), query languages, data models of next generation information systems, information retrieval and logic programming. He has published several articles on these topics in refereed journals and conferences. He also teaches logic programming at the Department of Computer and Information Sciences.