The design and implementation of a meaning driven data query language.

Epaminondas Kapetanios[1]
David Baer
Paul Groenewoud
P.Mueller
Department of Computer Science, Swiss Federal Institute of Technology, Zurich, Switzerland

[1]Epaminondas Kapetanios now works in the Harrow School of Computer Science, University of Westminster

The Design and Implementation of a Meaning Driven Data Query Language

E. Kapetanios, D. Baer, P. Groenewoud, P. Mueller
Dept. of Computer Science
Swiss Federal Institute of Technology
Zurich, Switzerland
kapetanios@inf.ethz.ch

Abstract

We present the design and implementation of a Meaning Driven Data Query Language - MDDQL - which aims at the construction of queries through system made suggestions of natural language based query terms for both scientific application domain terms and operator/operation ones. A query construction blackboard is used where query language terms are suggested to the user in its preferred natural language and in a name centered way, together with their connotation. This helps in understanding the meaning of the terms and/or operators or operations to be included in the query. Furthermore, the construction of the query turns out to be an incremental refinement of the query under construction through semantic constraints, where only those domain language terms and/or operators/operations are suggested which result into meaningful combinations of query terms as related to the scientific application domain semantics. Therefore, semantically meaningless queries can be prevented during the query construction. Such a semantics aware mechanism is not available in conventional database query languages such as SQL, where one is allowed to execute a query calculating, for example, the average of numerical data values whereas they represent the codes of categorical values. Moreover, no familiarity with the semantics of complex database schemes or interpretation of the symbols (names of classes/tables/attributes, value codes) underlying the storage model, as well as familiarity with the syntax of a database specific query language are needed by the end-user. The constructed query can be submitted to the MDDQL query interpretation and transformation engine, where the corresponding SQL-query is generated and delegated to a DBMS (e.g., Oracle, MS-Access, SQL-Server). Generation of SQL-statements addressing NF2 data models such as those provided by the object-relational Oracle DBMS is also enabled. The query result is presented in a table based form where all storage model symbols are interpreted and can be exported for the usage with statistical software packages (e.g., SPSS).

1. Motivation

Query languages as provided by database management systems, e.g., SQL, are mainly designed for programmers. Therefore, when end-users need to pose queries to a database, it is required that they have a substantial knowledge of syntax formalisms and the storage model semantics in order to formulate meaningful queries. The problem becomes more acute when scientific application domains or experiments are considered which address a large number of parameters (large database schemes). Moreover, the needs of formulating queries by using the familiar scientific terminology as expressed in their familiar natural language posed additional requirements for the design and implementation of the query language for accessing well-structured data from data repositories.

In order to meet these requirements, the implemented system aims at assisting the domain scientist to construct a query through system made suggestions of terms from the MDDQL vocabulary, which consists of both *application domain* and *operational terms*. Application domain terms are provided by the *application ontology*. The latter refers to an abstract model of the phenomena characterizing the particular application domain, i.e., *hospitalisation based treatment of myocardial infarction*, as well as the relevant concepts and the constraints on their use [12, 13]. Furthermore, the terms of the domain ontology are mapped to the *storage model symbols* standing for classes/tables, attributes and values.

All MDDQL vocabulary terms can be specified in words from more than one natural language. This is also applicable to their informal definition or connotation. The system made suggestions of terms, as provided by an *inference engine*, takes into consideration not only the semantic constraints as related to the application ontology but also to

Proceedings of the 14th International Conference on Scientific and Statistical Database Management (SSDBM'02)
1099-3371/02 $17.00 © 2002 **IEEE**

the operational terms. In other words, since no learning of a query language syntax is requested in order to construct an ad-hoc query, the incremental construction of the query takes place through end-user choices from those system inferred subsets of vocabulary terms which are semantically consistent with the set of terms already considered in the query. Distinction among *homonyms* such as *medication* is, therefore, enabled through the context of the term given either by its relationships with other terms or the underlying informal definition of the term.

This mechanism aims at assisting in constructing *meaningful queries* in terms of a) including only those well–understood terms - natural language words versus acronyms and codes - within the query and, therefore, e.g., only relevant relations, attributes and/or values at the storage model, b) applying those operations or operators which are consistent with the application domain semantics, e.g., only the *equals* operator would be suggested when a categorical value is considered within a restriction clause or the *average* operation will be excluded from suggestion when an attribute is classified as *categorical variable*, c) avoiding combination of mutually exclusive query terms such that they might lead to an empty set as a query result (unnecessary execution of query), d) providing a natural language based interpretation of the values as included in the query result.

The demonstration aims at showing how it is possible to construct queries by having the system driving the end-user to the construction of ad–hoc meaningful queries, i.e., queries which comply with the *application domain semantics*. Thereby, an object-relational database schema with approximately 120 attributes referring to *the treatment of acute myocardial infarctions* in Swiss hospitals will be used. The generation of the query result succeeds in that the constructed query is transformed at the server side into SQL-statements. Notice that *subqueries* addressing *nested tables* are also taken into account.

Given the mapping from the natural language words, in which the MDDQL query has been constructed, to acronyms and codes of the storage model, the same query result will be retrieved and sent to the client, even if the query has been constructed by using words from different natural languages. Presentation of the query result is given in a tuple-oriented form, where data items can be set-oriented values. The query result can be exported as an ASCII file with user specified delimiters such that the query result data can be imported into statistical software packages and/or front-end databases such as SPSS, Excel, MS-Access, etc. In such a case, multi-valued attributes are exported after having been flattened. All elements included in the query result can be viewed interchangeably, either by using the natural language based interpretations or the acronyms and codes as in the database.

Related work: The system does not make use of any diagrammatic presentation of conceptual schemes or any visual formalisms, and, therefore, contrasts with the approaches taken by visual query languages [2, 3, 7, 8, 9, 6] and systems (VQSs). They are mainly classified into *form-based* such as [9], *diagrammatic* such as [7] and *iconic* VQSs [8] according to the representation of the domain of interest for query formulation and/or for the query result. Some hybrid systems such as [6] have also been proposed. In all these high-level querying approaches, the application domain semantics based construction of a query, as described above, does not happen to be of major concern. In addition, the end–user is expected to make him/herself familiar with *visual formalisms* instead of *syntax formalisms*.

Semantics have been extensively used as a user guiding mechanism in interactive query formulation techniques. They are either database schema bound in order to provide incremental or associative query answering [15, 14], or they are ontology driven [11, 10, 1]. The goal of [15, 14] is to provide an end-user with context-sensitive assistance based on database modeling and probabilistic reasoning techniques combined with linguistic-based ones. To this extend, query formulation takes place in terms either of query completion of incomplete queries or suggestions of further predefined queries to reach a complex query goal [5, 4]. To this extent, no advanced semantics in terms of an advanced vocabulary or application ontology is user as a guiding mechanism for the construction of the query.

Instead, an ontology driven mechanism is provided by [11, 10, 1] which enables the clarification of the semantics of terms, such as in biology, to be used for the construction of the query. However, the main focus of the query construction technique is the exploration of taxonomies or classification structures rather than a querying mechanism capable of providing the expressive power as known by conventional database specific query languages. To this extent, usage of logical, comparison or statistical operators is not enabled. Moreover, semantic inconsistencies among terms cannot be prevented.

2. The MDDQL System Components

The system consists of the following components:

1. the *MDDQL query construction blackboard* (a snapshot is given by figure 1) used as the user interface, where the query is being assembled through user/system interaction in terms of requested suggestions of query terms and their signification as related to the specific application domain. The underlying data structure which gets manipulated through the user/system interaction on the blackboard and at the client site is the MDDQL query tree. The nodes of the

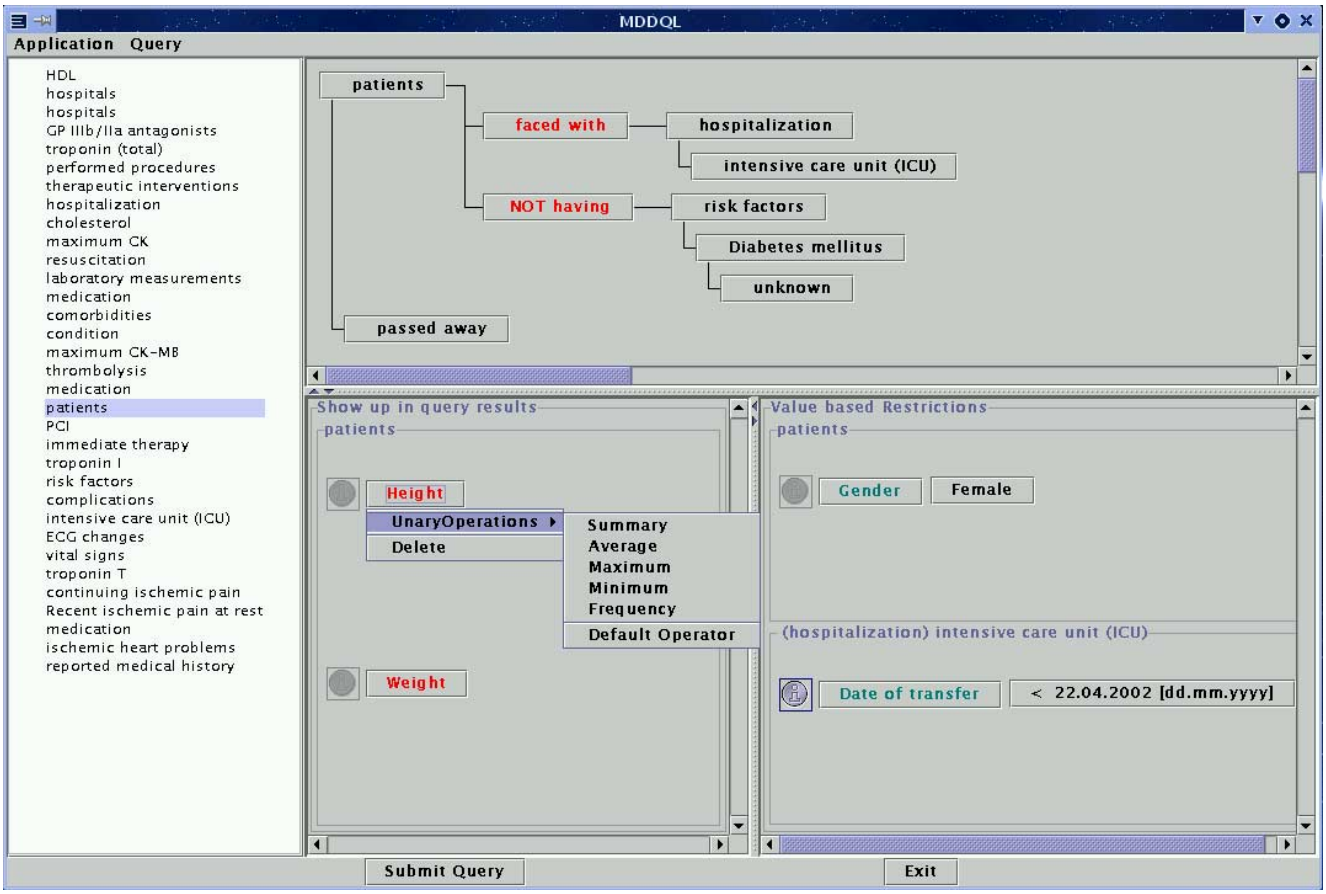

Figure 1. An example of a potential query

query tree are conceived as interconnected objects carrying on additional semantic knowledge needed by the interpretation/transformation component.

2. the *terminology base* which acts as the representation platform for the MDDQL query terms (vocabulary) together with their constraints on use which constitute the application ontology.

3. the *MDDQL inference engine* in order to respond to the user's requests for meaningful suggestions of query terms during query construction. The inference engine is contacted each time suggestions for the consideration of semantically consistent subsets of query terms are requested by the end-user in order to refine the query. The suggestions can be posed from selection menus which can be activated from each term already included within the query under construction. The inferences rely on both the semantics as represented by the *terminology base* and the current set of already considered query terms.

4. the *MDDQL query tree interpretation and transformation component*, where each submitted query in terms of an MDDQL query tree is transformed into an SQL-query.

The system relies on a three-tier system architecture. Assuming that the database resides at the *back–end layer*, components (2) and (4) are assigned to the *middle layer* whereas components (1) and (3) are assigned to the *front-end layer* (client site). However, a query construction session at the client site makes use of component (2) a copy of which is downloaded from the application server at the middle layer. Therefore, there is no network communication overhead between *inference engine* and *terminology server* during a query construction session.

Moreover, given this system architecture, there is a shift of the query construction logic from the middle layer to the clients. The application server deals only with the query transformation logic (component (4)). Given also that all system components are implemented in the *Java* programming language, the components (1) and (3) as needed for the

interactive query construction sessions can be installed as a software package on any client running under various operating systems such as Windows 98/NT/2000, Linux, Solaris, HP-AIX, etc.

Furthermore, maintenance of the *terminology base* is done at the application server independently of the interactive query construction software once installed at the client site. Any changes to the terminology space become immediately visible to all clients having installed the query construction components. Hence, the query vocabulary can be adapted dynamically according to changes of the application domain semantics.

2.1 An overview of the SQL generation algorithm

Since the submitted query takes the form of an MDDQL query tree which carries on all the semantic information needed for the mappings between natural language words and symbols to be used within the SQL statement, the generation algorithm has been implemented on the basis of traversing with a *depth–first* strategy the submitted MDDQL query tree. Thereby, the inference of the tokens and/or clauses to be put into the **select**, **from** and **where** slots relies a) upon the nature of the visited query nodes as well as the nature of the visited path, b) upon the nature of the corresponding storage model symbols, i.e., standing for a table, an attribute or attribute path (nested tables), a value or set of values.

Notice that the MDDQL query nodes are, mainly, classified as *Entity Set*, *Relationhip*, *Property* and *Value* nodes. Following constraints hold for the structure of an MDDQL query tree: an *Entity Set node* might have edges to either one or more *Entity Set* nodes, or one or more *Relationhip* nodes, or one or more *Property* nodes. A *Relationhip* node might have edges to one or more *Entity Set* nodes, or one or more *Property* nodes. A *Property* node might have edges to either one or more *Property* nodes, or to one or more *Value* nodes. Finally, a *Value* node might have edges to one or more *Value* nodes.

Currently, it is possible to infer JOIN operations with or without additional tables implementing a relationship. This also holds for tables used for classification structures. A distinction is made by the algorithm when categorical values are used for classification within the same table (no JOIN operation needed). Moreover, subqueries referring to nested tables such as those used for multi-valued attributes are also generated. This is crucial when *AND-connected* restrictions on values coming from nested tables are addressed.

Acknowledgments: We would like to thank Prof. Dr. M. Norrie and Prof. Dr. H. Hinterberger for their support to make this system reality.

References

[1] S. Bechhofer, R. Stevens, G. Ng, A. Jacoby, and C. Goble. Guiding the User: An Ontology Driven Interface. In N. W. Paton and T. Griffiths, editors, *Proc. User Interfaces to Data Intensive Systems (UIDIS99)*, pages 158–161, Edinburgh, September 1999. IEEE Press.

[2] J. Cardiff, T. Catarci, and G. Santucci. Semantic query processing in the VENUS environment. *International Journal of Cooperative Information Systems*, 6(2):151–192, June 1997.

[3] T. Catarci, M. Costabile, S. Levialdi, and C. Batini. Visual query systems for databases: A survey. *Journal of Visual Languages and Computing*, 8(2):215–260, April 1997.

[4] W. Chu and Q. Chen. A Structured Approach for Cooperative Query Answering. *IEEE Transactions on Knowledge and Data Engineering*, 6(5):738–749, October 1994.

[5] W. W. Chu, H. Yang, K. Chiang, M. Minock, G. Chow, and C. Larson. CoBase: A scalable and extensible cooperative information system. *Journal of Intelligent Information Systems*, 1996.

[6] L. Cinque, S. Levialdi, and F. Ferloni. An expert visual query system. *Journal of Visual Languages and Computing*, 2:101–113, 1991.

[7] Y. Dennebouy, M. Anderson, A. Auddino, Y. Dupont, E. Fontana, M. Gentile, and S. Spaccapietra. SUPER: Visual interfaces for object + relationship data models. *Journal of Visual Languages and Computing*, 6:74–99, 1995.

[8] A. Massari, S. Pavani, L. Saladini, and P. Chrysanthis. QBI: Query by icons. In *Proc. of the ACM SIGMOD Conf. on Management of Data*, page 477, San Jose, USA, 1995. ACM Press.

[9] G. Ozsoyoglou and H. Wang. Example-based graphical database query languages. *COMPUTER*, 26(5):25–38, May 1993.

[10] N. Paton, R. Stevens, P. Baker, C. Goble, S. Bechhofer, and A. Brass. Query Processing in the TAMBIS Bioinformatics Source Integration System. In *Proc. 11th Int. Conf. on Scientific and Statistical Databases (SSDBM)*, pages 138–147 1999. IEEE Press, 1999.

[11] R. Stevens, P. Baker, S. Bechhofer, G. Ng, A. Jacoby, N. Paton, and C. Goble. TAMBIS: Transparent Access to Multiple Bioinformatics Information Sources. *Bioinformatics*, 16(2):184–186, 2000.

[12] R. Studer, R. Benjamins, and D. Fensel. Knowledge Engineering: Principles and Methods. *DKE*, 25(1-2):161–197, 1998.

[13] M. Uschold and M. Grueninger. Ontologies: Principles, Methods and Applications. *Knowledge Engineering Review*, 2, 1996.

[14] G. Zhang. *Interactive Query Formulation Techniques for Databases*. PhD thesis, University of California, Los Angeles, 1998.

[15] G. Zhang, W. W. Chu, F. Meng, and G. Kong. Query Formulation from High-Level Concepts for Relational Databases. In N. Paton and T. Griffiths, editors, *Proc. User Interfaces to Data Intensive Systems, UIDIS 99*, pages 64–74, Edinburgh, Scotland, September 1999. IEEE Computer Society Press.