Simplifying Syntactic and Semantic Parsing of NL Based Queries in Advanced Application Domains

E. Kapetanios and D. Baer and P. Groenewoud

Dept. of Computer Science
Swiss Federal Institute of Technology (ETH)
ETH-Zentrum, CH-8092 Zurich, Switzerland
email: kapetanios@inf.ethz.ch

Abstract: The paper aims at presenting a natural (sub)language based querying approach (MDDQL) for SQL (relational, object-relational) databases, which relies on an ontology driven, interactive query construction mechanism. This guides the user to the construction of queries that are semantically compliant with the application domain semantics. To this extent, syntactic and semantic parsing of a query is done implicitly, during the query construction, rather than syntactically and semantically parsing the query after its formulation. Given also that the vocabulary terms are represented as objects having properties and not as simple natural language words, it is possible to cope with the intentional meaning of terms as well as with homonyms, i.e., the same word meaning different things, during the query construction. In addition, it is possible to express the query terms in a different natural language without changing or providing another syntactic and semantic parsing mechanism. Therefore, the generated SQL-statement reflects not only the application domain semantics but is also identically inferred from multi-lingual queries. This querying approach is currently applied to a Swiss national registry, which consists of, at the moment, twelve data repositories of patients' records, but it can also be applied to any scientific or technical domain with an advanced and hardly understood terminology.

Keywords: Natural Languages, Query Languages, Ontologies, Semantic parsing, SQL, Information Retrieval, Automata.

1 Introduction

Querying databases through Natural Language (NL) based query languages - we exclude *keywords based* querying - has always attracted research and development efforts in order to ease access to and increase understandability of data repositories [DL96, BBFU96, BM99, CCS00]. Syntactic and semantic parsing of NL based queries, however, turns out to be tedious [AG99, MW01], especially when complex or advanced terminologies are used like those found in scientific and technical application domains. Mostly, they rely on parsing a query in terms of constructing a query tree which is compliant with the underlying NL syntactic and semantic rules.

In addition, constructing a query presupposes that the user is familiar with the terminology of the application domain. In other words, the user needs to know not only the typography of words but also their meaning. It would be impossible or, at least, overoptimistic to assume that one makes him/herself familiar with the full range of the scientific or technical domain vocabulary. Therefore, the more advanced or complex terminologies become, the more prone to syntactic and semantic mistakes becomes the process of NL based query construction. Furthermore, the lack of awareness of the full range of the scientific or technical terminology might also leave large parts of a database schema unexploited.

On the other side, since semantic parsing mostly refers to the NL based semantic properties of words within the query [Kle56, Knu68] and not to the application domain semantics itself [UG96], it is possible to construct a syntactically and semantically correct query, in terms of the NL related semantics, which is still meaningless [Ull62]. Moreover, enabling querying of databases through multi-lingual vocabularies increases the complexity of syntactic and semantic parsing of queries [Zhd02], since identical data should be accessed from queries which have been parsed according to the underlying NL syntactic/semantic rules. This turns out to be tedious, since either a single parser needs to be aware of various NL related syntactic/semantic rules or different parsers need to be implemented.

Finally, as far as the semantic parsing process itself is concerned, it is tedious to take into consideration the intentional meaning of words [MSPK96], regardless the underlying NL, in order to resolve ambiguities such as those characterising homonyms, even if only a sublanguage is concerned.

In our approach, we rely on a natural (sub)language based querying approach (MDDQL) for querying SQL (relational, object-relational) databases, which is an ontology driven, interactive query construction mechanism. This guides the user through an incremental construction of queries, which is accomplished by suggesting terms being semantically compliant with the application domain semantics. To this extent, syntactic and semantic parsing of a query is done implicitly, during the query construction, rather than syntactically and semantically parsing the query after its formulation. Therefore, a query is not prone to syntactic or typographic mistakes, where still the application domain semantics are respected. Additionally, all available data might be exploited, since the whole database schema as reflected by the vocabulary can be explored.

Given also that the vocabulary terms are represented as objects [Nor00] having properties and not as simple words, it is possible to cope with the intentional meaning of terms as well as with ambiguities as stated by homonyms, i.e., the same word meaning different things, during the query construction. In addition, it is possible to express the query terms in a different natural language without increasing the complexity of the syntactic and semantic parsing mechanism. Furthermore, the generated SQL-statement is not only compliant with the application domain semantics, but is also identically inferred from multi-lingual queries.

This querying approach is currently applied to a Swiss national registry, which consists of, at the moment, twelve data repositories of patients' records, but it can also be applied to any scientific or technical domain with an advanced and hardly understood terminology.

Organisation of the paper: Section 2 gives an insight into the organisational, ontological structure of the query language vocabulary. Section 3 refers to the terms suggestion mechanism as a kind of inference mechanism operating upon the ontological structure of the vocabulary. Section 4 gives an overview of the MDDQL transformation and SQL-statement generation logic. A conclusion summarizes what has been presented in the paper. An example of a query construction and SQL-statement generation is presented throughout all sections of the paper in order to illustrate the approach.

2 The query language vocabulary

The vocabulary of MDDQL is a *set of terms* which are conceived as objects. The vocabulary terms refer to database elements such as attributes, values, etc. This frames-based representation enables the representation of terms by taking into consideration *properties* of terms rather than simply its name. Therefore, the *name* of the term as represented by a natural language word becomes a property (slot) of the term. Further slots enable the capture of the intentional meaning of the term as expressed by a given annotation, the assignment of *term unique* or *object identifiers*, images illustrating a term, etc.

All vocabulary terms are connected to each other in a cyclic graph by using the *term unique identifiers* and not the *names* of the terms. This provides a terminological space where all application domain terms with their constraints on use are expressed. The links used for the interconnection of terms either lead to *hierarchical* structures within the same class of terms or to *attributive* structures, i.e., among different classes of terms. We mainly distinguish among five classes of terms: *concepts*, *relationships*, *properties*, *values* and *operations*.

Links which constitute hierarchical structures of terms are classified according to the semantics of the linkage such as *is-a*, *part-of*, *constitutes-of*, etc. Attributive links connect, for example, terms which belong to *concepts* with terms which belong to *properties* or those which belong to *relationships*. These kind of links might specify the potential properties and relationships which circumscribe a particular concept.

Accordingly, there are, for example, terms which have been assigned the words *Immediate therapy*, *Thrombolysis*, *Medication* and are classified as *concepts*. *Medication* as an instance of the class *concepts*, however, is defined more than once in the same vocabulary. Therefore, consideration of terms is bound to the underlying context as given by the intentional meaning of terms. This is defined in terms of slots such as `annotation`, `condition of measurements`, `measurement unit`, etc., as well as in terms of the hierarchical and attributive links connecting that particular term to other terms in the neighborhood. The links, in our example, make clear the distinction between *Medication as kind of Immediate Therapy* and *Medication prescribed to Discharged Patients*.

On the other hand, terms having assigned the names *Date of thrombolysis*, *Reasoning for denial of thrombolysis* and classified as properties are connected with the term named *Thrombolysis*, which is classified as concept, through attributive links. Similarly, the term named *given to*, which is a member to the class of terms *relationships*, is further connected

with the term named *patients* with an attributive link too. Attributive links also connect the terms named *Thrombolysis* and *Medication* with the terms named *PCI preferred* and *Aspirin*, respectively, which, in turn, are members of the class of terms *values*.

Given this vocabulary organisational structure, it is possible to express *homonyms*, i.e., two terms having assigned the same name, or even replace the assigned words as expressed in a particular natural language such as *English* with the words as expressed in another natural language such as *German*, without changing the organisational or ontological structure of the vocabulary (ease of maintenance).

Despite the fact that *Ontological Engineering* is still a relatively immature discipline, some methodologies and criteria of analyzing them can be found in [Lop99]. To this extent, the ontological engineering approach taken so far is closer to the SENSUS-based methodology, which aims at structuring and representing of conceptual structures to be used as a dictionary for natural language processing [KL94, KCH+95, SS01, SS02]. However, one of the major differences is that conceptual structures are organized around *names of terms* and not around *objects* as it is the case for the $MDDQL$ vocabulary.

Moreover, hierarchical and attributive links as well as slots can be assigned constraints which make their validity relative to particular conditions. Therefore, query terms become relative to a context as defined by the terms already included in the intented query and the neighboured terms in the vocabulary. For example, constraints might have an impact on the relativity of attributes and value domains according to the nature of the target database. The term named *Troponin I*, classified as *properties*, is not valid, if the target database for the intended query is the *Triemli* hospital, since there is no such measurement taking place at this hospital.

The vocabulary is completed by terms which belong to the class of terms *operations*. This class includes all terms standing for *comparison*, *logical*, or *statistical operators*. Constraints also hold on the validity of these terms, since not all sets of operations make sense, when a particular context of terms is given, which are already included in the query. This is mainly defined by their role as arguments for statistical or comparison/logical operations, respectively, operators.

A detailed description of the inference of the validity of terms relying upon the assigned constraints outlies the scope of this paper. In the following, we give an overall description of the query construction mechanism, which is based upon the suggestion of meaningful terms. It relies upon the organisation or ontological structure of the MDDQL vocabulary and the terms already taken into consideration for the intended query.

3 The query construction paradigm

The major idea behind the MDDQL query construction mechanism has been the avoidance of an *a-posteriori* syntactic and semantic parsing of a query. Following the conventional approach of having, at first, the query typed by the user and, subsequently, parsed syntactically and semantically by the system has the following drawbacks: a) the increased complexity of the syntactic and semantic parsing of a query in a multi-lingual user community,

b) unexploitation of the intentional meaning of both application domain and operational terms, c) no exploitation of the full range of scientific or technical vocabularies, given that end-users are not fully familiar with the domain, and, therefore, query formulation is prone to syntactic and semantic mistakes.

In order to alleviate query construction in complex domains and still having, as far as possible, a meaningful query, i.e., a query which reflects the application domain semantics, the MDDQL query construction mechanism relies upon suggestion of terms through a human-computer interaction mechanism rather than the conventional approach of first *typing* and then *parsing*. Three sub-goals, however, had to be considered.

- The constructed query should be reflected by a *high-level query tree* including all terms of the intended query.
- Only those terms should be considered which are compliant with the application domain semantics. Therefore, the suggestion of terms is performed on the basis of semantic inferences which rely on the organisational or ontological structure of the vocabulary.
- Generation of the corresponding SQL statement by traversing the *high-level query tree*.

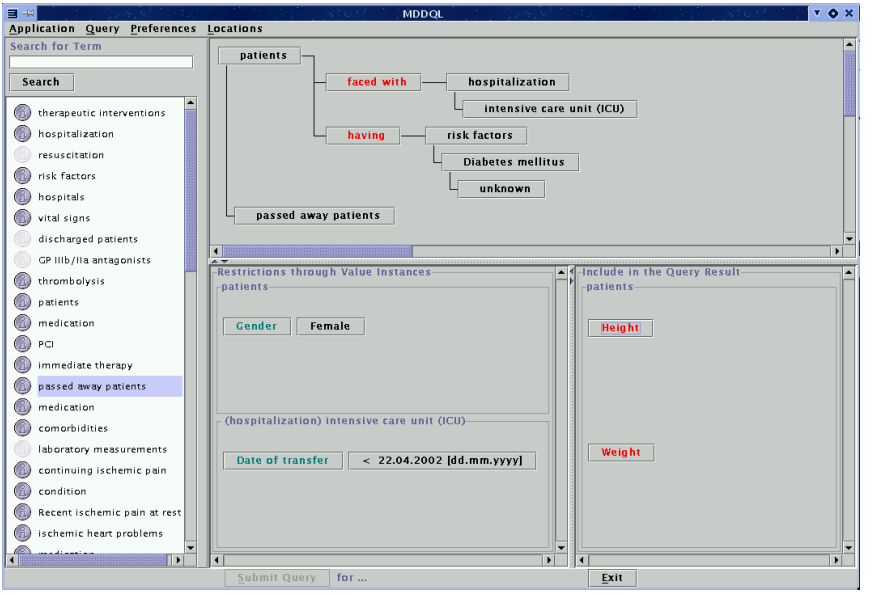

Figure 1: A snapshot of the MDDQL query construction blackboard

The query construction takes place on a blackboard as depicted in figure 1. All terms appear on the blackboard by using the values (words) as assigned to the slot *name* of the

vocabulary terms. The user has the possibility of starting the query construction process with a term as selected from a list of initial terms which appears on the left panel. The list includes those terms which correspond to major concepts characterizing the application domain. Alternatively, a searching mechanism is also available in order to locate the requested term within the given vocabulary, in order to begin the query construction.

Since searching is done on the basis of a *non-unique name assumption*, i.e., on the words assigned as values to the slot of terms *name*, all relevant terms are included in the answer together with their context as defined by the linkages and the frame structure of terms. Note that in order to define the neighbored terms, the ontological structure of the vocabulary is being navigated in terms of the term unique identifiers (TUI's) and not the words themselves.

The three panels on the right is a means of expressing the intended query. The panel at the top includes only terms which belong to the classes of terms *concepts* and *relationships*. Therein, it is only possible to express queries in terms of *subject-predication* associations in a sub-language. For example, *patients having received immediate therapy* is a query expression with the term named *patients* in the role of the *subject* and the terms named $\{$*having received, immediate therapy*$\}$ in the role of the *predication*. The terms appearing on the two panels at the bottom express the values based restrictions and what has to be included in the query result (projection), respectively.

Since construction of the query takes place in terms of suggestion of terms, we distinguished between two basic categories of suggestions: a) those which refine the intended query and, consequently, the query result, b) those which decide upon inclusion of properties or attributes within the query results, i.e., a metaphor for the *projection* operation. All kinds of suggestions can be requested by drop-down menus activated on each particular application domain term of the intended query. For instance, having selected the term named *Patients*, which is classified as *concepts* term and placed on the top panel, there are following possible kinds of restrictions to be suggested in order to further refine the intended query:

- *Set of predications* such as *faced with hospitalization*, *having risk factors*, *having received immediate therapy*, which are constructed by following all ougoing edges (pair of terms) in the ontological structure of the vocabulary, as connected to that particular term by attributive links with the first term classified as *relationships*.
- *Specializations* of generic terms such as the terms named *discharged* or *passed away* which are constructed from outgoing hierarchical links.
- *Value based restrictions* as posed on characteristic properties of the affected term classified either as *concepts* or *relationships*, such as *height*, *weight* or *gender*, which are constructed from outgoing attributive links providing connections to terms classified as *properties* within the ontological structure of the vocabulary.

The *projection* operation, i.e., attributes to be included in the query result, is simulated by selecting one or more attributes from the suggested set of terms, which are classified as instances of the class *properties* and are compliant with the ontological structure of the

vocabulary. Outgoing attributive links are allowed for terms being members of either the class *concepts* or the class *relationships*.

3.1 Reflecting the query on a high-level MDDQL query tree

The construction of the query, however, is reflected by the synchronous construction or manipulation of a high-level MDDQL query tree construct, which resides in the working memory of the query construction blackboard. The query tree is defined in terms of *query term nodes*. A *node* is conceived as a thin version of the frame based definition of the corresponding term from the ontological structure of the vocabulary. This means that they carry on only the values which are necessary for the MDDQL to SQL transformation (figure 2).

For example, the values of the slot `annotation` are not included in the query tree, since they are only useful for the selection of query terms. To the contrary, the values of the slot `symbol` - represented in angle brackets - are always included, since they refer to the corresponding texture of word as used by the storage (database) model, e.g., names of tables, attributes, codes for values, etc. (more details in 4.1.1).

Assuming that the intended query would be `the dates of thrombolysis for discharged patients, where reasoning for denial of thrombolysis is that PCI has been preferred`, as given by free text, the incremental query construction is reflected by the four query tree manipulation stages as depicted in figure 2. At stage (A), the query tree reflects the refinement of *patients* by the selected predication *having received immediate therapy*. At stage (B), the query tree reflects the further refinement of the intended query through the selected specialisation term of *discharged*, i.e., restriction of the query result to those patients who have been discharged.

At stage (C), the query tree reflects a further refinement by focussing on *thrombolysis* as a *kind-of immediate therapy*, which has been selected from suggested specialization terms. Finally, stage (D) reflects the query tree having taken the final form before submission, where the terms named *date*, *reasoning for denial*, *PCI preferred* will be interpreted by the MDDQL to SQL transformation algorithm: the first as a projection, the two others as value based restriction.

Since it is semantically meaningless to ask for the *date of thrombolysis* in conjunction with the restriction that there must be a reason for *denial of thrombolysis*, it is known to the ontological structure of the vocabulary that this pair of terms is mutually exclusive. Therefore, the intended query, respectively, the query tree will be rejected as meaningless prior to its submission for transformation and execution.

To this extent, a meaningful assignment of operators or operations is also taken into account in the intended query. Therefore, only semantically consistent, for example, operators are suggested according to the nature of attributes. This also applies to the class of unary operations such as *average*, *maximum*, *minimum*, etc., which can be suggested as applicable functions for properties such as *Height* but not to *Gender*, since the latter has been classified as a `Categorical Variable` in the ontological structure of the

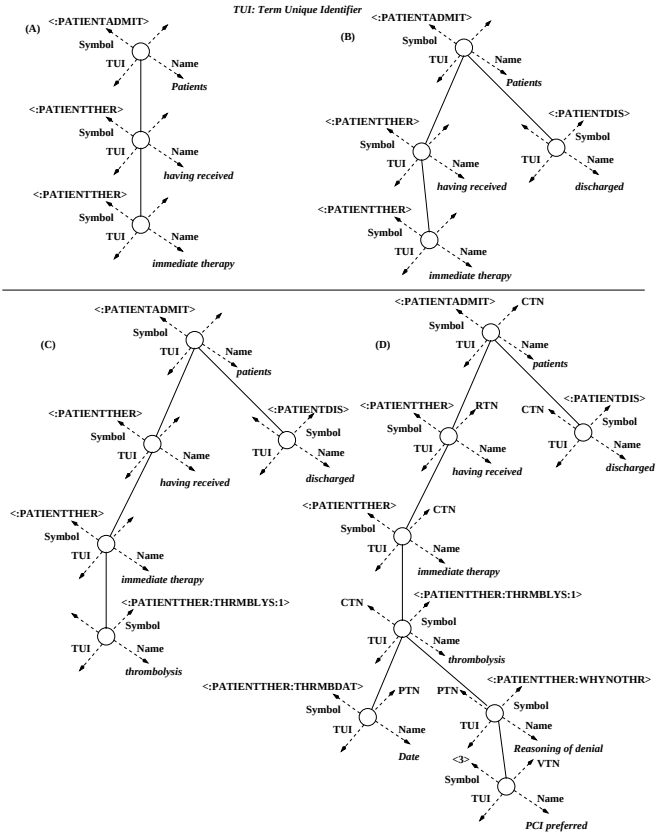

Figure 2: Incremental MDDQL query tree construction and manipulation

vocabulary.

The default logical and comparison operators holding for the constructed query tree, if no other semantically compliant operator has been selected, are the `AND` and the `equals` operators, respectively. For the sake of simplicity, we do not cover all aspects of operator/operation assignment to the query tree nodes.

In the following, only the major aspects of the MDDQL query transformation logic is presented. We intend to give an insight into the transformation logic rather than presenting all algorithmic details and, therefore, outrange the scope of the paper.

4 MDDQL query interpretation and transformation

The transformation logic of an MDDQL query tree towards an SQL-statement is based on filling an *SELECT FROM WHERE* pattern conceived as an assembly of the SELECT, FROM and WHERE clauses, while traversing the submitted MDDQL query tree in a *depth-first* strategy. The tokens to be written as well as their destination, i.e., SELECT, FROM or WHERE clause, to which they belong, are determined by

- the notation of the corresponding storage model symbols (see also values of the slot `symbol` as depicted in figure 2), which are elements of the description of the storage or database model and values,
- the nature of the submitted MDDQL query tree,
- the meta-data which refers to the description of the logical database schema.

In the following subsection 4.1, we briefly refer to each of these SQL-statement generation ingredients, before embarking into the overview of the transformation logic (subsection 4.2).

4.1 The ingredients

4.1.1 The storage model symbols (SMS)

All database model and value elements, which constitute an implemented database, are conceived as storage model symbols. They are referred as assemblies of one or more names of relations (tables), attributes, values, as well as the database itself, however, in the following interpretation order as defined by the notation

$$< [database] : relation : attribute : value >$$

This notation indicates, implicitly, the assignment and role of an element within the storage model with `database` being optional. For example, the term named *thrombolysis*, in the ontological structure of the vocabulary, is realized by the SMS $<: PATIENTTHER : THRMBLYS : 1 >$. This indicates the fact that *thrombolysis* has been realized as the value 1 of the attribute **THRMBLYS** that is part of the definition of the table **PATIENTTHER**.

Given that abstract data types can also be referred by SMSs too, an **attribute** in the SMS notation can be described as *dot separated* list of attributes, whereas a **value** can be described by a list of values. For example, the term named *instances of regular medication* is represented by the SMS $<: HISTPATIENT : HISTMED.ITEM\_NAME >$, where **ITEM_NAME** is an attribute of the nested table **HISTRMED**. Accordingly, the term named **smoker** is not directly represented by one single value and, therefore, the corresponding

SMS takes the form $<: HISTRF : SMOKE : [1,2] >$, where $[1,2]$ stands for various categories of *smoker* such as *ex-smoker*, and *current smoker*.

The SMS contents not only become parts of the contents of the SELECT, FROM and WHERE clauses, since they provide the tokens for the generated SQL-query, but they also partly determine additional sequences of characters or restrictions to be considered by one or more clauses. For example, since we are interested in only those `patients having received thrombolysis as a kind of immediate therapy`, as reflected by the query tree (D), figure 2, the query result will be restricted not only by the inferred *equijoin* between the relations **PATIENTADMIT** and **PATIENTTHER** (see below for more details), but also by a value based restriction in the WHERE clause, such as $THRMBLYS = 1$. Instead, if the SMS $<: PATIENTTHRMBLYS >$ would hold indicating that a separate relation would have been used in order to realize the concept *thrombolysis*, then an *equijoin* between the relations **PATIENTTHER** and **PATIENTTHRMBLYS** is inferred.

4.1.2 The nature of the submitted query tree

The major contribution to the form of the generated SQL query statement, however, comes from the nature of the submitted MDDQL query tree, in terms of a) the semantic classification of the query tree nodes, b) their arrangement within the query tree.

Given that each term fro which a query tree node has been derived is mainly classified as *concepts*, *relationships*, *properties*, *values*, *operators* - just to name a few - in the ontological structure of the vocabulary, the potential structure of a submitted MDDQL query tree is specified in the following in terms of constraints that the arrangement of nodes in the MDDQL query tree should satisfy. The query tree nodes will be referred in terms of their ontological classification as abbreviated in the following: a) Concept Term Node **CTN**, from terms which are instances of the class *concepts*, b) Relationship Term Node **RTN**, from terms which are instances of the class *relationships*, c) Property Term Node **PTN**, from terms which are instances of the class *properties*, d) Value Term Node **VTN**, from terms which are instances of the class *values*.

The MDDQL query tree specification constraints holding are: a) the *root* is always a CTN, b) a VTN has an incoming edge from a node which is either a VTN or a PTN, c) a PTN has an incoming edge from a node which is either a PTN or CTN or RTN, d) if CTN is not *root*, then a CTN has an incoming edge from a node which is either a CTN or RTN.

An example of ontological classification of the query term nodes is given by the query tree (D) as depicted in figure 2. The classification acronyms are given as values of the corresponding slot of a query term node.

4.1.3 The meta-data for the logical database schema

The description of the logical database schema is given by an XML based notation. This aims at describing the definition of tables in terms of assigned attributes, primary and foreign keys, data types, etc. A subset of such a description is given in the following:

```
< Relationname = "PATIENTADMIT" >
                < Attributename = "PATIENT_ID" dt : type = "NUMBER" primary = "primary"
                        < RefTable relation = "PATIENTTHER" attribute = "PATIENT_ID" / >
                < /Attribute
                < Attributename = "HOSPREC_ID" dt : type = "VARCHAR2" primary = "primary" >
                        < RefTable relation = "CONDITION" attribute = "HOSPREC_ID" / >
                        < RefTable relation = "VITALSIGNS" attribute = "HOSPREC_ID" / >
                < /Attribute >
                < Attributename = "SUBMISSION_DATE" dt : type = "DATE" / >
                < Attributename = "BIRTHDAT" dt : type = "DATE" / >
                < Attributename = "SEX" dt : type = "VARCHAR2" / >
                < Attributename = "ADMISDAT" dt : type = "DATE" / >
                < Attributename = "HEIGHT" dt : type = "NUMBER" / >
                < Attributename = "WEIGHT" dt : type = "NUMBER" / >
                < Attributename = "PATINSURANCE" dt : type = "VARCHAR2" / >
                < Attributename = "PATTRANSFER" dt : type = "VARCHAR2" / >
< /Relation >
```

This kind of meta-data is required in order to infer the names of the attributes over which two tables should be joined as well as the types of the attributes involved in the WHERE clause (value based restrictions) and the SELECT clause (presentation of the query result). For instance, if the given type of an attribute in the WHERE clause is VARCHAR2, then the corresponding value will be enclosed in single quotes.

4.2 The cooking

It is not our intention to present the specification of the automaton underlying the implementation of the transformation logic. We would like, however, to give a short description of the transformation logic.

Since the MDDQL query tree is traversed in a depth-first strategy, the SQL-statement generation is conceived as an automaton having as an initial state $q_0 = \{S_0, F_0, W_0\}$, where $S_0 \equiv F_0 \equiv W_0 \equiv \{\}$, with S, F, W representing the SELECT, FROM, WHERE parts of the SQL-statement.

The combination of classifications of the MDDQL query term nodes, which constitute a visited edge, determine what to write and in which part of the SQL-statement. In other words, each visited edge might cause a move from one state to another such that $q_i = \{S_i, F_i, W_i\}$, with, for example, $F_i \neq \{\}$.

For example, if a $< PTN, VTN >$ edge is being visited, this causes the generation of a value based restriction for the WHERE clause such as **WHYNOTHR = '1'** (query tree (D) in figure 2). If an edge is being visited having a PTN as a leaf node, then the corresponding symbol is written into the SELECT clause such as **THRMBDAT** (query tree (D), figure 2).

Accordingly, inference of *equijoin* operations becomes a matter of detecting such operations during the traversal of the submitted MDDQL query tree by the query transformation algorithm. This, mainly, depends (a) on the combination of ETN and RTN query term nodes which belong to the *visited query tree (sub)path* as well as (b) the contents of the corresponding SMSs. Thus inferred *equijoin* operations take also into account how a relationship has been implemented. An *equijoin* operation is enabled for both:

- **additional relation (table)** such as those realizing **N:M** relationships between entity sets,

- **no additional relation (table)** such as those realizing **1:1** or **1:N** relationhips between entity sets.

In both cases, additional *equijoin* based restrictions need to be considered for the WHERE clause. The conditional restriction is built upon the input provided by the XML metadata specification of the logical database schema, as briefly presented above. For instance, given a visited edge $< CTN, CTN >$ having two different names of relations in their SMSs, such as **PATIENTADMIT** and **PATIENTDIS** (query tree (D), figure 2), an *equijoin* operation is inferred in terms of their primary/foreign keys. In other words, the SQL-statement under generation has been moved into a state where **PATIENTADMIT PATIENTADMIT** and **PATIENTDIS PATIENTDIS** are written into the FROM clause (the second token stands for the variable), with the restriction **PATIENTADMIT.PATIENT_ID = PATIENTDIS.PATIENT_ID** added to the WHERE clause.

Similarly, given the combination $< CTN, RTN, CTN >$ in a visited path having **PATIENTADMIT**, **PATIENTTHER** and **PATIENTTHER** as corresponding SMSs, only the **PATIENTADMIT PATIENTADMIT**, **PATIENTTHER PATIENTTHER** are considered for the FROM clause, respectively, only the *equijoin* restriction **PATIENTADMIT.PATIENT_ID = PATIENTTHER.PATIENT_ID** is considered for the WHERE clause, since the relationship has been implemented by using only two tables. This is inferred from the identical SMS $<: PATIENTTHER >$ as assigned to both the RTN and one of the connected ETNs).

It is also possible to consider nested tables as well as operators, which are assigned to particular query tree nodes. A thorough description of the automaton outlies the scope of this paper. The final state $S_{q_f}, F_{q_f}, W_{q_f}$ of the automaton reflects the generated SQL-statement and is reached when the MDDQL query tree has been fully traversed. Following our example (query tree (D) from figure 2), the constructed query will take the form:

SELECT PATIENTTHER.THRMBDAT
FROM PATIENTADMIT PATIENTADMIT, PATIENTTHER PATIENTTHER,
PATIENTDIS PATIENTDIS
WHERE PATIENTADMIT.PATIENT_ID = PATIENTDIS.PATIENT_ID **AND**
PATIENTADMIT.PATIENT_ID = PATIENTTHER.PATIENT_ID **AND**
PATIENTTHER.THRMBLYS = '1' **AND**
PATIENTTHER.WHYNOTHR = '3'

The same SQL-statement would have been generated, if words from a different natural language were assigned as values for naming terms in the ontological structure of the vocabulary and, consequently, in the submitted query tree nodes. To this extent, the MDDQL transformation / SQL-statement generation logic remains unchanged. Finally, the elements (attributes and values) of the query result are presented to the user in such a way that they are interpretable in the natural language in which the query has been posed.

5 Conclusion

We presented a querying approach which simplifies syntactic and semantic parsing of queries, especially in multi-lingual, advanced application domains such as scientific or technical ones. The approach relies on an ontology driven, interactive query construction mechanism, which leads to the interactive construction and manipulation of a high-level query tree rather than constructing a syntactic and semantic query tree while parsing a completely constructed query. This turns out to be cumbersome, since the complexity of syntactic and semantic parsing increases considerably or ineffective, since constructing a query with natural language terms presupposes a considerable knowledge of the scientific or technical domain terminology. This is also strengthened by the fact that semantic parsing by also taking into consideration the intentional meaning of terms is tedious or even impossible. The high-level query tree gets transformed into SQL-statements with respect to the underlying application domain semantics. This querying approach, however, can similarly be applied for the generation of any database specific query statements, which are not bound to the SQL syntax.

References

[AG99] Vincenzo Ambriola and Vincenzo Gervasi. Experiences with Domain-Based Parsing of Natural Language Requirements. In *4th International Conference on Applications of Natural Language to Information Systems*, Klagenfurt, Austria, 1999. IOS Press.

[BBFU96] J. Bernauer, A. Benneke, A. Fuesechi, and M. Urban. Structured Data Entry for Medical Records and Reports. In *Second International Workshop on Applications of Natural Language to Information Systems*, Amsterdam, The Netherlands, 1996. IOS Press.

[BM99] Batrice Bouchou and Denis Maurel. Natural Language Database Query System. In *4th International Conference on Applications of Natural Language to Information Systems*, Klagenfurt, Austria, 1999. IOS Press.

[CCS00] Roger H.L. Chiang, Chua Eng Huang Cecil, and Veda C. Storey. A Smart Web Query Engine for Semantic Retrieval of Web Data and its Application to E-trading. In *5th International Conference on Applications of Natural Language to Information Systems*, pages 215–226, Versailles, France, 2000. IOS Press.

[DL96] F. Dinenberg and D. Levin. Natural Language Interfaces for Environmental Data Bases. In *Second International Workshop on Applications of Natural Language to Information Systems*, Amsterdam, The Netherlands, 1996. IOS Press.

[KCH+95] K. Knight, I. Chancer, M. Haines, V. Hatzivassiloglou, E.-H. Hovy, Masayo Iida, Steve K. Luk, Richard Whitney, and Kenji Yamada. Filling Knowledge Gaps in a Broad-Coverage Machine Translation System. In *Proc. of IJCAI95*, pages 1390–1397, Montreal, Canada, 1995.

[KL94] K. Knight and S. Luck. Building a Large Knowledge Base for Machine Translation. In *Proc. of the AAAI-94*, pages 779–784, Seattle, USA, 1994.

[Kle56] S. C Kleene. *Automata studies*, chapter Representation of events in nerve nets and finite automata, pages 3–42. Princeton Univ. Press, Princeton, N.J., 1956.

[Knu68] Donald E. Knuth. Semantics of Context-Free Languages. In *Mathematical Systems Theory*, volume 2, pages 127–145. 1968.

[Lop99] Mario Fernadez Lopez. Overview of Methodologies for Building Ontologies. In *Proc. of the IJCAI-99 Workshop on Ontologies and Problem-Solving Methods (KRR5)*, Stockholm, Sweden, August 1999.

[MSPK96] M.-S.Jeon, S.-Y. Park, and M.-S. Kim. Extraction of Exact Meaning Using a Keyfact Concept System. In *Second International Workshop on Applications of Natural Language to Information Systems*, Amsterdam, The Netherlands, 1996. IOS Press.

[MW01] Markus Mittendorfer and Werner Winiwarter. Experiments with the Use of Syntactic Analysis in Information Retrieval. In *6th International Conference on Applications of Natural Language to Information Systems*, pages 37–44, Madrid, Spain, 2001. IOS Press.

[Nor00] M. C. Norrie. Advances in Object-Oriented Data Modelling. In M. Papazoglou, S. Spaccapietra, and Z. Tari, editors, *Object-Oriented Data Modelling Themes*, pages 1–18. MIT Press, 2000.

[SS01] Vijayan Sugumaran and Veda C. Storey. Creating and Managing Domain Ontologies for Database Design. In *6th International Conference on Applications of Natural Language to Information Systems*, Madrid, Spain, 2001. IOS Press.

[SS02] Vijayan Sugumaran and Veda C. Storey. An Ontology Based Framework for Generating and Improving DB Design. In *7th International Workshop on Applications of Natural Language to Information Systems*, pages 1–12, Stockholm, Sweden, 2002. Springer Verlag.

[UG96] M Uschold and M. Grueninger. Ontologies: Principles, Methods and Applications. *Knowledge Engineering Review*, 2, 1996.

[Ull62] S. Ullman. *Semantics - An Introduction to the Science of Meaning*. Blackwell, Oxford, 1962.

[Zhd02] Anna V. Zhdanova. Automatic Identification of European Languages. In *7th International Conference on Applications of Natural Language to Information Systems*, pages 76–84, Stockholm, Sweden, 2002. IOS Press.