Simplifying syntactic and semantic parsing of NL-based queries in advanced application domains.

Epaminondas Kapetanios *
David Baer
Department of Computer Science, Swiss Federal Institute of Technology

Paul Groenewoud
University of Applied Sciences, Switzerland

* Epaminondas Kapetanios now works in the Harrow School of Computer Science, University of Westminster

This is an electronic version of an article published in Data & Knowledge Engineering, 55 (1). pp. 38-58, October 2005. The definitive version in Data & Knowledge Engineering is available online at:

http://www.sciencedirect.com/science/journal/0169023X

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

E. Kapetanios [a], D. Baer [a], P. Groenewoud [b]

[a] *Dept. of Computer Science,*
Swiss Federal Institute of Technology, Zurich (ETHZ),
Hirschengraben 84, CH-8092 Zurich, Switzerland,
email: kapetanios@inf.ethz.ch, dbaer@acm.org

[b] *University of Applied Sciences,*
Oberseestr. 10, CH-1475, Rapperswil, Switzerland
email: pgroenew@hsr.ch

Abstract

The paper presents a high level query language (MDDQL) for databases, which relies on an ontology driven automaton. This is simulated by the human-computer interaction mode for the query construction process, which is driven by an inference engine operating upon a frames based ontology description. Therefore, given that the query construction process implicitly leads to the contemporary construction of high level query trees prior to submission of the query for transformation and execution to a semantic middle-ware, syntactic and semantic parsing of a query with conventional techniques, i.e., after completion of its formulation, becomes obsolete. To this extent, only, as meaningful as possible, queries can be constructed at a low typing, learning, syntactic and semantic parsing effort and regardless the preferred natural (sub)language. From a linguistics point o view, it turns out that the query construction mechanism can easily be adapted and work with families of natural languages, which underlie another type order such as *Subject-Object-Verb* as opposed to the typical *Subject-Verb-Object* type order, which underlie most European languages. The query construction mechanism has been proved as practical in advanced application domains, such as those provided by medical applications, with an advanced and hardly understood terminology for naive users and the public.

Key words:
Natural Languages, Query Languages, Ontologies, Semantic Web, Databases, Information Retrieval, Automata Theory.

1 Introduction

1.1 Background

Querying databases through Natural Language (NL) based query interfaces - we exclude *keywords based* querying - has always attracted research and development efforts in order to ease access to and increase understandability of the full potential of information as provided by large data repositories [1–5]. Syntactic and semantic parsing of NL based queries, however, turns out to be tedious [6,7], especially when complex or advanced terminologies are used like those found in scientific and technical application domains. Mostly, this kind of query parsing relies in constructing a query tree which is compliant with the underlying NL–based syntactic and semantic rules, which are different as they depend on the preferred NL.

In addition, constructing a query presupposes that the user is familiar with the terminology itself as well as the semantic relationships among terms of the application domain. In other words, the user needs to know not only the orthography of words and their intentional meaning, as expressed in a particular natural (sub)language, but also how they relate to each other. It would be impossible or, at least, too optimistic to assume that one makes itself familiar with the full range of the scientific or technical domain vocabulary, in both terms, spelling and meaning.

Therefore, the more advanced or complex vocabularies become, the more prone to syntactic and semantic mistakes becomes the process of NL–based query construction and parsing. This is also strengthened by multi–lingual user communities, where different parsing techniques need to be considered and implemented. Consequently, either the complexity of the parsing technique increases dramatically or large parts of the information potential in databases for scientific or engineering applications remain unexplored and cannot become part of exploratory querying.

Moreover, it is still possible to construct a syntactically and semantically correct query, in terms of the NL–based semantics, however, the query might be still meaningless [8], since it might not reflect *real–world* semantics. This is due to the fact that semantic parsing mostly refers to the NL–based grammatical roles of words or structure of the query [9,10], i.e., which is the *subject* or *noun phrase*, the *verb*, the *object*, the *connectives*, etc., and not to the application domain semantics itself [11] as expressed by some kind of ontological considerations. A query saying *All cars aged more than 40 years, which have been infected by AIDS* is grammatically correct as far as the grammatical roles (noun phrase, relative clause, verb, etc.) of the participating words and phrases have been correctly recognized by the parsing technique.

Complexity of query parsing and transformation is further increased, since, regardless the preferred natural language, the same *query extensional semantics*, in terms

of tuples received from the databases as an answer to the query, are expected to hold. To this extent, either the same parser needs to consider various NL–specific syntactic and semantic rules, or different parsers need to be enabled according to the number of the preferred natural languages. Finally, it is tedious or even impossible to integrate into the parsing technique the intentional meaning of words [12], in order, for example, to resolve ambiguities such as those characterizing homonyms, i.e., terms with the same spelling but different meanings, even within natural (sub)languages.

In order to overcome the difficulties of parsing natural language due to real–world semantics, even in a restricted form such as that given by a query language, several query construction approaches have been taken in the recent past, which rely on a diagrammatic presentation of some conceptual model [13,14] reflecting some kind of real–world semantics. These vary in their expressiveness from logical, mostly relational, models to high level ones such as Extended Entity–Relationship (EER) and Object Modelling Technique (OMT) diagrams. These query construction techniques, however, are restricted in their expressiveness, given their limitations of reflecting advanced real–world semantics such as constraints, exception handling, etc., which might lead to the construction of meaningless queries. Additionally, they are bound to visual formalisms, which introduce semantic complexity.

In an attempt to provide a query construction facility by reflecting more advanced real–world semantics, ontology driven mechanisms have been suggested for construction of queries such as those found in [15–17]. This approach enables the construction of queries around the semantics of terms as given by their classification and taxonomic relationships. The main focus, however, has been the exploration of taxonomies or classification structures rather than the specification of a high-level query language with the expressive power as partly known by conventional database specific query languages, e.g., expression of logical, comparison or statistical operators, arithmetic expressions, etc.

This is due to the fact that ontology descriptions rely on languages, which are heavily influenced by a family of class–based (concept–based) knowledge representation formalisms known as Description Logics [18]. The reasoning services, therefore, are targeted at the explication of the relationship between the ontology language syntax and the intended model(s) of a domain via *model theoretic semantics* as a formalization of the meaning of the ontology description language. Consequently, the reasoning services are not targeted at the construction of meaningful queries, including operators and operations.

1.2 Our approach

On the contrary, our intention and approach, as presented in this paper, has been to help users construct meaningful queries, while still making use of terms in an arbitrary natural language. The reasoning services during querying construction are, therefore, targeted at the structure and formulation of the query from a linguistic point of view. For several years, however, the trend in natural language research has been oriented toward the elaboration of huge linguistic dictionaries and ontologies [19–22], including relations between concepts and common sense. The exploitation of such dictionaries fulfilled some ”understanding” requirements, however, with quite sophisticated parsing techniques [23,24].

In order to simplify the query formulation process and parsing, while preserving the same *extensional semantics* for multi–lingual queries, we restricted the query language vocabulary to a (sub)language as provided by the ontology. This includes not only application domain terms, but also operators and operations, in order to extend the expressiveness of the query language beyond taxonomies and classifications. On the other hand, given that the query formulation is driven by the ontology, the construction of the query becomes a matter of adding meaningful sets of all kinds of terms as suggested by the system. The suggestion of sets of terms, however, is driven by reasoning services, which take into consideration

- the context of terms as circumscribed by the real–world semantics and expressed by the ontology,
- the user environment,
- the partly constructed query.

To this extent, the query construction process resembles the moves among potential states as specified by an automaton with the reasoning services determining the potential moves. It is this ontology driven automaton, which specifies MDDQL as a query language.

From a linguistic point of view, the automaton also *prescribes* the *word order type* which underlie any constructed query. Currently, queries are constructed by following the *Subject–[Verb–Object]* word order type, which characterizes 42 percent of all natural languages [25]. Moreover, according to the *parametric theory* of natural languages, as stated in [25], where *parameters* and not *words* are conceived to be the *atoms* of a language, it is easier to construct a meaningful query, regardless the preferred natural language, assuming that the same word order type holds.

To this extent, the automaton shifts the grammatical structure of queries from *words* to *word order types*. This, in turn, simplifies the parsing technique in cross–lingual environments in terms of simply replacing the words from one natural language to another in the ontology, while preserving the same automaton, i.e., similar reasoning services during query construction.

The simplification of query construction and parsing is also strengthened by the fact that the query under construction is reflected, in real time, on a high level conceptual query tree, which is being manipulated, i.e., changing the contents of the query tree (adding/deleting nodes or sub-trees) or contents of nodes themselves, according to changes, e.g., adding or deleting terms, in the query under construction. Given also that the query tree is constructed by selecting terms out of *suggested* meaningful sets of terms, it turns out that syntactic parsing of the query becomes obsolete, since the query is not prone to syntactic or orthographic mistakes.

Furthermore, semantic parsing is also alleviated due to the context aware suggestions of terms. *Context of terms* is defined by their *interrelationships* as expressed by the ontology as well as by their *intentional meaning* as expressed by their annotations. The latter enables disambiguation of meaning of terms during query construction, not only if it is difficult to grasp the meaning of a scientific or technical term by simply looking at a word or name, but also when the same word is used with different meanings in the query language vocabulary.

Organization of the paper: Section 2 gives an insight into the considerations of the ontology based vocabulary description of the query language. A partial ontology description of the vocabulary exemplifies these considerations. Section 3 refers to the reasoning services underlying the query construction process due to the context aware suggestions of terms in conjunction with particular NL word order types as known by linguistic theories. Section 4 illustrates the MDDQL transformation logic of the high level query trees, which underlie the submitted query, on the example of on–the–fly generation of SQL statements. A conclusion summarizes what has been presented in the paper.

2 The query language vocabulary

The vocabulary of MDDQL is described by an ontology language, which is close to the OWL [1] [26] new ontology language for the Semantic Web, as developed by the World Wide Web Consortium (W3C) Web Ontology Working Group. Given that we put the emphasis on readability and general ease of use as important considerations of an ontology language to become a platform of creating and modifying ontologies, a surface syntax based on the *frames* paradigm has been chosen. Frames group together information about each class or instance and, therefore, make ontologies easier to read and understand, particularly for those who are not familiar - or do not want to become - with (Description) Logics [26]. The frames paradigm has

[1] http://www.w3.org/2001/sw/WebOnt/

been used in a number of well known knowledge representation systems including Protege–2000 [27] and the OKBC [2] knowledge model.

In frames based languages, each class or instance is described by a frame. The frame includes the name of the class, identifies the more general class or classes that it specializes, and lists a set of ”slots”. A slot, in turn, may consist of a property–value pair or a constraint on the values (individuals or data values) that act as a slot ”filler”. In addition, frames can be used in order to describe *properties* as having range and domain constraints, specializing more general properties or having inverse property relationships.

In accordance with these knowledge representation issues, all available terms (application domain, operators, operations) are defined in the ontology language as being *class*, *property*, *value* and *instance* elements. Given that we use separate frames in order to represent terms for *properties*, e.g., *age*, and *value* elements, e.g., [20–120], frames become ”fillers” of some other frame slots. *Property* frames also represent relationships among agents, which are expressed by verbs in natural language, e.g., *received, admitted to, etc.*, having a *subject* and an *object* in the roles of agents, which, in turn, are represented by *class* or *instance* elements. These are known in OWL as *object properties* in order to distinguish them from those properties, which relate classes or instances to data-type values, e.g., *title*, *age*, and are known in OWL as *data type properties*.

To this extent, *slots* are reserved as modelling constructs of the ontology description language. They are, roughly speaking, classified as *attributive*, *hierarchical* or *membership* slots, according to their roles as modelling constructs. *Hierarchical* slots provide the modelling constructs *subClassOf* , *unionOf* , *intersectionOf* , *disjointWith*, etc., which are used in order to specialize or define abstract classes (terms).

Moreover, *subClassOf* slots are classified as *isKindOf* , *isPartOf* and *constituteSOf* slots. *Attributive* slots are used in order to relate *property* elements to *class* or *instance* elements. In this case, they can be further classified either as *objectPropertySlots*, when they refer to *object properties*, or as *datatypePropertySlots*, when they refer to *data type properties*. Furthermore, they are also used to relate *value* elements to *property* elements.

In the following, we give an example of a partial ontology description as taken from a real–world case study. Slots are referred as *slot–constraints* with their classification indicated by a semi–colon.

class *o1*.Patients
    slot–constraint:objectPropertySlot *o15*.admitted to
    slot–constraint:objectPropertySlot *o16*.have received

[2] http://ontolingua.stanford.edu/okbc

slot–constraint:objectPropertySlot *o17.*have been transferred from
slot–constraint:datatypePropertySlot *o19.*age
slot–constraint:datatypePropertySlot *o20.*height
slot–constraint:datatypePropertySlot *o21.*insurance
slot–constraint:memberOf *m10.*Classes *meta–modelling level*

. . . . . .

class *o10.*Discharged patients
slot–constraint:subClassOf:isKindOf *o1.*Patients
slot–constraint:objectPropertySlot *o33.*have been transferred to
slot–constraint:objectPropertySlot *o34.*to receive
slot–constraint:datatypePropertySlot *o39.*Date of discharge
slot–constraint:memberOf *m10.*Classes *meta–modelling level*

. . . . . .

property *o15.*admitted to
slot–constraint:subPropertyOf *o100.*transferred to
slot–constraint:objectPropertySlot:domain *o1.*Patients
slot–constraint:objectPropertySlot:range *o2.*Hospitals
slot–constraint:datatypePropertySlot *o40.*admission date
slot–constraint:memberOf *m11.*Properties *meta–modelling level*

. . . . . .

property *o16.*have received
slot–constraint:objectPropertySlot:domain *o1.*Patients
slot–constraint:objectPropertySlot:range *o50.*Medication
slot–constraint:memberOf *m11.*Properties *meta–modelling level*

. . . . . .

property *o34.*to receive
slot–constraint:objectPropertySlot:domain *o10.*Discharged patients
slot–constraint:objectPropertySlot:range *o30.*Medication
slot–constraint:objectPropertySlot:isInverseProperty *o45.*recommended to
slot–constraint:memberOf *m11.*Properties *meta–modelling level*

. . . . . .

property *o19.*age
slot–constraint:hasValue *o110.*[20–120]
slot–constraint:memberOf *m112.*Numerical variables *meta–modelling level*

. . . . . .

property *o21.*insurance
slot–constraint:hasValue *o120.*public, private
slot–constraint:memberOf *m111.*Categorical variables *meta–modelling level*

. . . . . .

instance *o99.*Aspirin
memberOf *o50.*Medication

.....

instance *o100.*Aspirin
memberOf *o30.*Medication

.....

In contrast with OWL, cross–references among the ontology elements are established through IDs rather than *names*, for two purposes: (a) in order to express *homonyms*, i.e., two terms having assigned the same name, however, with different extensional semantics, e.g, *medication*, within the same ontology name space without introducing artificial names, (b) in order to have the query construction reasoning services working regardless the preferred natural (sub)language in which the query is being constructed. In our example of the partial ontology above, we made use of terms in *English* for the sake of simplicity and readability.

Since the appearance of a term to the user takes place through an assigned name, e.g., *aspirin, discharged patients*, which becomes the value of a slot for all ontology elements, we feel the need to distinguish between slots as modelling constructs for the description of the *interrelationships* among the ontology elements and *inherited slots* from meta–modelling frames. These frames stand for the description of ontology elements themselves, which, in turn, are conceived as being instances of these meta–modelling frames.

In an object–oriented terminology, the meta–modelling frames are represented by meta–classes where all ontology elements are instances. In other words, the *class*, *property*, *value, instance* elements are instances themselves of the classes for all *classes, properties, values, instances*, respectively. The common slots, such as *term unique identifier, name, annotation, measurement unit, operationalRole, synonyms, validity preconditions*, etc., provide a ”deeper” structure to the description of each ontology element itself, and, therefore, a deeper understanding of the terms.

For example, understanding of the term *medication* might be strengthened by the value as assigned to the slot *annotation*. It is this *intentional semantics*, which alleviate the task of disambiguation of meaning of advanced or homonym terms, in addition to the ontology modelling constructs. *Preconditions* might also have an impact on the meaning of all kinds of ontology description elements, within the same ontology space, according, to the target database. For example, the *instance* element *Total Troponin* is only valid, if the target database for the intended query is supposed to be the hospital named *Triemlispital*, since there is no such measurement taking place at other hospitals. The latter explicates that an *instance element* is bound to a particular location.

An extension of our partial ontology description example through the *meta–modelling* elements by using the same modelling constructs is given in the following:

class *m1*.Vocabulary
    slot–constraint:datatypePropertySlot *m19*.term unique identifi er
    slot–constraint:datatypePropertySlot *m20*.name
    slot–constraint:datatypePropertySlot *m21*.annotation
    slot–constraint:datatypePropertySlot *m22*.precondition

. . . . . .
class *m10.*Classes
slot–constraint:subClassOf:isPartOf *m1.*Vocabulary
. . . . . .
class *m11.*Properties
slot–constraint:subClassOf:isPartOf *m1.*Vocabulary
. . . . . .
class *m12.*Values
slot–constraint:subClassOf:isPartOf *m1.*Vocabulary
. . . . . .
class *m13.*Instances
slot–constraint:subClassOf:isPartOf *m1.*Vocabulary
. . . . . .
class *m111.*Categorical variables
slot–constraint:subClassOf:isKindOf *m11.*Properties
. . . . . .
class *m112.*Numerical variables
slot–constraint:subClassOf:isKindOf *m11.*Properties
slot–constraint:datatypePropertySlot *m25.*Measurement Unit
. . . . . .

This kind of ontological abstraction also allows the expression of preconditions, which hold between collections of ontology description elements, such as those holding between *comparison operators* or *operations* and *properties*, since not all (sub)sets of operators or operations make sense to be suggested for inclusion in the query. This strongly depends on the *context of terms* of that part of the vocabulary, which refers to the application domain. For instance, given that we have the following description at the meta–modelling level,

class *m5.*Operators
slot–constraint:subClassOf:isPartOf *m1.*Vocabulary
. . . . . .
class *m51.*Comparison Operators
slot–constraint:subClassOf:isKindOf *m5.*Operators
. . . . . .
class *m52.*Unary Operations
slot–constraint:subClassOf:isKindOf *m5.*Operators
. . . . . .

we could express preconditions saying that *Unary Operations* can be applied to properties classified as *Numerical variables*, whereas the subset of *Unary Operations* with instances $\{$*Mean, Variance, Deviation*$\}$ cannot be applied to properties classified as *Categorical variables*. Similarly, the subset of *Comparison Operators*

with instances $\{>, <, >=, <=, between\}$ makes sense to be considered together with properties classified as *Numerical variables* only.

Summarizing, the *context of query language terms* is defined in terms of (a) the membership to the meta–modelling frames, (b) their semantic interrelationships, (c) contextualization of the ontology itself as expressed by the holding preconditions. The reasoning services for guiding the user to the construction of meaningful queries in cross–lingual communities need to take into consideration this *context*. In the following, we give a short overview of the mechanism underlying the reasoning services by emphasizing its role as natural language constructor based on particular *word order types*.

3 The query construction through reasoning services

3.1 A linguistics point of view

The major goal of the design and specification of MDDQL as a high level query language has been the simplification of construction and parsing of meaningful queries, with an advanced query language vocabulary, especially when more than one natural (sub)language can be used. In order to meet this goal, the following considerations have been taken into account:

(1) User guidance to the intended query.
(2) Syntactic and semantic parsing of a query is done *implicitly* during query construction.
(3) The constructed query should be reflected on a *high-level query tree*.

Since the real–world semantics are provided by the ontology as described in the previous section, the user guidance during the interactive query construction relies upon *reasoning services*, which take into consideration both the *real–world semantics* and the *current state of the intended query*. To this extent, the query construction process resembles the definition of an *automaton* by having the reasoning services to determine the potential states to move on, i.e., potential refinements of the intended query. In other words, it is like having an automaton underlying the specification of a query language, where *the real–world semantics guide the syntax rather than having the syntax guiding the semantics of a language*.

The major challenge, however, behind the specification and realization of the reasoning services and the automaton as an interaction mode has been the construction of meaningful queries in cross–lingual environments. Therefore, the syntax of the constructed query should reflect a particular *word order type*, which characterizes the preferred natural language. It has been shown by linguists [25] that, despite the

combinatorial explosion in differences at various levels of natural languages, there are some principles and commonalities which underlie all known human natural languages. It is, therefore, not the *words*, which are considered as the *atoms of a language*, but rather *parameters* which lead to classification of natural languages according to some *word order type*.

A rough classification of natural languages according to some basic word order types and their distribution is given in the following:

- Subject–[Verb–Object], 42 percent, for example, English, German, Indonesian
- Subject–[Object–Verb], 45 percent, for example, Japanese, Turkish
- Verb–Subject–Object, 9 percent, for example, Zapotec, Welsch
- etc.

Currently, the reasoning services and the automaton are targeted at constructing natural (sub)language based queries according to the Subject–[Verb–Object] word order type, which characterizes 42 percent of natural languages. To this extent, simply replacing the *names* of the ontology elements from one natural language to another, e.g., replacing the English terms with the German ones, without any changes to the reasoning services and automaton specification, still produces queries with a meaningful syntax in 42 percent of natural languages.

3.2 From the ontology to natural language

However, in order to get a query constructed according to this particular word order type, we need to consider the modelling constructs of the ontology from a linguistic point of view. Given that *Subject*, *Object* and *Verb* are rather *noun* and *verb phrases*, respectively, than simple words, the query construction guidance mechanism mainly distinguishes among the various *slots* and their roles in the ontology description.

Roughly speaking, a noun phrase can be constructed by elements, which are connected through all kinds of slots, which are not classified as *objectPropertySlots*. According to the semantics as assigned to this kind of slots (section 2), *objectPropertySlots* provide the connectives between the noun phrases, standing either for the *Subject* or the *Object*, and the *Verb phrase*, which is mainly provided by the corresponding *property* element.

For instance, a legitimate query would have been ”The age and weight of discharged patients admitted to hospitals with location Zurich”, with ”age and weight of discharged patients” being the *Subject*, ”admitted to” being the *Verb*, and ”hospitals with location Zurich” being the *Object* parts of the query. This is due to the fact that, for example, *age* and *weight* are *data type properties* of *patients* and, consequently, of *discharged patients*, given that there is a slot constraint *subClas-*

sOf:isKindOf holding between *patients* and *discharged patients* (section 2). In addition, there are slot constraints classified as *objectPropertySlot* holding in the ontology between *admitted to* and the classes *patients, hospitals*.

The construction of this query, however, would have been taken place incrementally due to the suggestions as made by the reasoning services. For example, given that *[discharged patients]* has been the term at an initial state, potential subsequent states would have been

- *[discharged patients, admitted to, hospitals]*,
- *[discharged patients, to receive, medication]*,
- *[discharged patients,* {*age, height*}*, admitted to, hospitals]*,

etc. The decision to which state to move on is taken by the user, each time she/he makes a choice among the suggested terms by the reasoning services. In addition, term selection is also bound to intentional semantics in terms of having a look at the annotation of vocabulary terms prior to inclusion. This leverages the choice of correctly defined terms, in case that the word alone does not suffice to disambiguate the meaning of a term.

Since the suggestions are made by taking into consideration the ontological context of terms and the current state of the query, states like *[discharged patients, received, hospitals]* or *[discharged patients,* {*location*}*, admitted to, hospitals]* will never appear in a query. Furthermore, given that all elements in the ontology, which are known as *classes*, can constitute an *initial state*, queries being in a state like *[medication, recommended to, discharged patients]* are possible, however, a query being in a state *[medication, recommended to, discharged patients, to receive, medication]*, i.e., building a cycle, would not be possible, since *to receive* and *recommended to* are known as *inverse properties* (section 2) and, therefore, *[to receive, medication]* will not be suggested as a potential move.

Examples of further semantically compliant moves among query states are guaranteed by examining the modelling constructs among *classes*, such as *disjointWith*. A query state such as *[discharged patients,* {*age, height*}*, passed away patients, admitted to, hospitals]* is not possible, since *discharged patients* and *passed away patients* are known to be *disjoint*. This would have been equivalent to a query like ”The age and height of discharged patients, *who have passed away*, as admitted to hospitals...”, which is meaningless.

The reasoning services have recently been extended by including operators or operations. The consideration of the *OR* logical operator in the query state above would have turned the query state to a permissible one, since a query like ”The age and height of discharged *OR* passed away patients, as admitted to hospitals...” is a meaningful one. On the other side, inclusion of operators or operations would change the state of the query and, therefore, they are also subject to the reasoning services for meaningful suggestions of terms.

For instance, comparison operators, which apply only to categorical variables, are excluded from suggestions in conjunction with *properties*, which are known as being instances of *categorical variables* (see also the partial ontology description in section 2). Therefore, a query state like *[discharged patients*, {*age*, *height*}, {insurance >= 'p'}, *admitted to*, *hospitals]* will not be possible. Additionally, statistical operations such as *variance*, or *mean value* are only permissible, if the affected *property* is known to be a member of *numerical variables*.

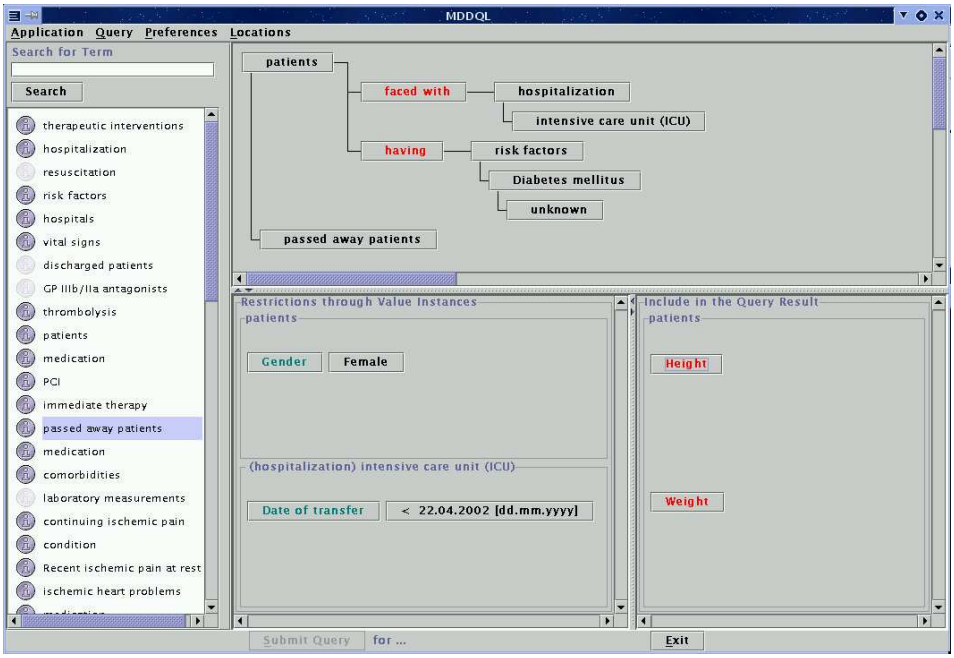

Fig. 1. A snapshot of the MDDQL query construction blackboard

The query construction takes place on a blackboard, as depicted in figure 1, which acts as the interaction means between user and machine. All terms appear on the blackboard by using the values, which are assigned to the slot *name* of the ontology elements, in the preferred natural (sub)language. 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. Alternatively, a search mechanism is also available in order to locate the requested term within the given vocabulary, in order to begin the query construction with. If an ontology element happens to have the same name as a value, such as *medication*, the search results are presented in conjunction with the term neighbourhood in order to help the user to disambiguate the term.

A full description and specification of the reasoning services as well as a mathematical definition of the ontology driven MDDQL automaton lies out the scope of this paper. It is worth mentioning, however, that the reasoning services rely upon traversing of the graph, which reflects the ontology description in its *frames* like knowledge representation. When imported into main memory, it constitutes an object–oriented semantic network upon which the reasoning services are performed

in terms of mainly looking for semantically consistent nodes at the neighbourhood of a particular node, which is already part of the intended query.

Currently, the neighbourhood is partly defined by adhering the nodes, which can be reached from a particular node standing for a *class* or *instance* through *object-PropertySlots* to other *classes* or *instances*, without changing the directionality of the traversed nodes. This enables a direct mapping to the *[Verb–Object]* adherence as foreseen for the *Subject–[Verb–Object]* word order type. However, reversing the directionality of the traversed nodes of this kind would cause the adherence of *[Object–Verb]* subtypes to the *Subject*, which, in turn results into the new word order type *Subject–[Object–Verb]*, which underlies the principles of natural language construction for another 45 percent of human languages such as Japanese, Turkish, etc.

3.3 Reflecting the query on a high level query tree

Regardless the word order type, according to which queries are formulated, the construction of the query is reflected by the synchronous construction or manipulation of a high-level query tree as an abstract data type construct, which resides in the working memory of the query construction blackboard. This reflection simplifies parsing and transformation of the constructed query toward database specific query languages, despite the potential huge range of applicable natural (sub)languages. It is this query tree which gets traversed in order to generate the database specific query language, as illustrated in the next section.

The high level 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 description of the vocabulary. This means that they carry on only those values or ”fillers”, which are necessary for the transformation toward database specific query languages. For example, the *annotation* slots are not included in the query tree nodes, since they are only useful for the construction of the query by providing explanation of terms. On the contrary, the *memberOf* slots are always included, since they refer to the role of nodes in terms of meta–modelling elements.

The structure of a potential high level query tree underlies the following constraints:

- The root of the query tree is always a *Class* or an *Instance* term node.
- A *Class* or *Instance* term node might have as children other *Class*, *Instance* or *Property* term nodes.
- A *Datatype Property* term node might have as children other *Datatype Property* term nodes or *Value* term nodes.
- An *Object Property* term node, i.e., relationship between two *agents*, MUST have children, which are *Classes* or *Instance* term nodes.

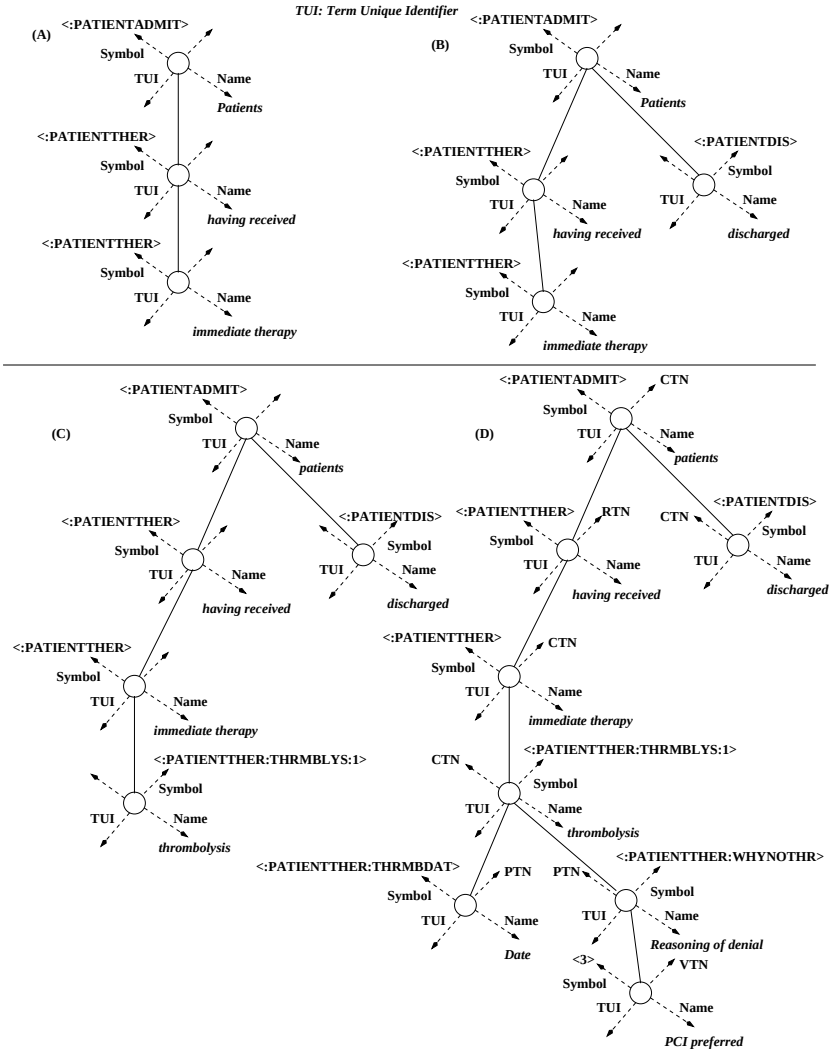

Fig. 2. Query construction states as reflected on a high level query tree

- An *Object Property* term node, i.e., relationship between two *agents*, might have as children *Property* term nodes.
- A *Value* term node might have as children only *Value* nodes.

Presumably, the intended query would be: *date of thrombolysis for discharged patients with reasoning for denial of thrombolysis "PCI has been preferred"*. The incremental query construction is reflected by the four query tree states as depicted in figure 2. At state (A), the query tree reflects the refinement of *patients* by the selected predication *have received immediate therapy*, where the query node *have received* is classified as an *Object Property* and the nodes *patients* and *immediate therapy* are classified as *classes*. At state (B), the query tree reflects the further refinement of the intended query term node *discharged*, i.e., restriction of the query result to those patients who have been discharged.

At state (C), the query tree reflects a further refinement by focusing on *thrombolysis*

as a *kind-of immediate therapy*, which has been selected from suggested specialization terms. Finally, state (D) reflects a *meaningless* query tree, since it semantically violates the constraint that both *date of thrombolysis* and *denial of thrombolysis* as parts of the intended query are not permissible. In other words, it would not be meaningful to ask for *date of thrombolysis* in a query, which has as reasoning for denial of thrombolysis *PCI has been preferred* as part of an AND–connected restriction clause. Therefore, it is expected that, in these cases, thrombolysis has not been performed at all and, correspondingly, construction as well as submission for transformation and execution of such a query tree will not be possible.

This also applies to the class of unary operations such as *average*, *maximum*, *minimum*, etc., which can be suggested as applicable functions for *datatype properties* being members of *Numerical variables*, such as *Height*, but not for *Gender*, since the latter has been classified as a *Categorical Variable* in the ontology based description of the vocabulary. Comparison operators and unary operations are part of the query node description in the high level query tree. Operators and operations which apply to more than one argument, they are assigned to query tree nodes which are roots of, at least, binary (sub)trees.

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. In the following, we give an insight into the transformation logic rather than presenting all algorithmic details and, therefore, outrange the scope of the paper.

4 High level query tree interpretation and transformation

The transformation logic of an MDDQL high level query tree toward a database specific query language relies upon a generative grammar, which is specified as an automaton having as an initial state the root of the submitted high level query tree. Subsequent states are determined by traversing the query tree nodes in a *depth-first* strategy. Therefore, each state is defined of all currently visited nodes or subgraph. According to the roles and contents of the query term nodes, it is determined which parts of the database specific query statement should be generated.

For instance, generation of an SQL-statement is based on filling an $\{$*SELECT, FROM, WHERE*$\}$ pattern conceived as an assembly of the SELECT, FROM and WHERE clauses. The tokens, however, to be written out as well as their destination, i.e., SELECT, FROM or WHERE clause, are determined by

- the mapping between the ontology elements as appear on the query tree nodes and the database implementation symbols,
- the nature of the submitted MDDQL query tree,

- the meta–data which refers to the description of the logical database schema.

Since the nature of the submitted high level query tree has been covered in section 3.3, in the following subsection 4.1, we briefly refer to the two other factors, mapping of ontology elements and meta–data for database logical schema description, as ingredients for the generation of, for example, SQL-statements, before embarking into the explication of the transformation logic (subsection 4.2) on the same example.

4.1 The ingredients

4.1.1 The storage model symbols (SMS)

The ontology elements to be found on the submitted high level query tree correspond to one or more *storage model symbols*, which are defined as assemblies of database model and value elements such as relations (tables), attributes, values, as well as the database itself. SMS's are constructed according to the notation

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

This notation indicates, implicitly, the containment as well as interpretation of the storage model element with *database name* being optional. For example, the term named *thrombolysis* (figure 2) in the ontology description of the vocabulary, is mapped into 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 complex data types need to be referred by SMS's 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 mapping, however, between *term unique identifiers* of ontology elements and SMS's is provided either by the ontology description or by a *mediation* layer during query transformation. Reference to this mapping at an ontology description level is done by including the slot *symbol* for all elements of the query language vocabulary. To this extent, the corresponding SMS is implicitly selected during the query construction and need not be retrieved or queried at the mediation level. This works accordingly when simple databases are targeted (see also example with query states as depicted by figure 2). In more sophisticated environments, however, where collections of databases with semantic heterogeneity are being targeted, a mapping of ontology elements to one or more SMS's at the *mediation* level is preferred.

The SMS contents not only provide the parts of the contents of the SELECT, FROM and WHERE clauses in the generated SQL statement, but they also have an impact on the final form of the generated SQL statement by determining additional clauses to be considered. For example, since we might be interested in only those *discharged patients having received thrombolysis as a kind of immediate therapy*, as reflected by the query tree state (C) (figure 2), the query result will be restricted not only by the inferred *equi–join* 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 *equi–join* between the relations **PATIENTTHER** and **PATIENTTHRMBLYS** is inferred.

4.1.2 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. 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 transformation of the high level query tree

It is not our intention to present the full specification of the automaton underlying the implementation of the transformation logic. We would like, however, to give a short overview of the transformation logic. Since the MDDQL high level query tree is traversed in a depth-first strategy, the SQL statement generation is conceived by the automaton having 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, assuming that an $< PTN, VTN >$ edge, with PTN being a *datatype property node* and VTN being a *value term node*, is visited on the query tree state (D) in figure 2, and that the corresponding meaningless query would have been permitted, this would have caused the generation of a value based restriction for the WHERE clause such as **WHYNOTHR = '1'**. Within the same hypothetical query tree, if an edge is visited having a PTN, e.g., *date of thrombolysis* as a leaf node, then the corresponding symbol **THRMBDAT** is written into the SELECT clause as candidate for projection.

Accordingly, inference of *equi–join* operations becomes a matter of detecting such operations during the traversal of the submitted MDDQL high level query tree by the query transformation algorithm. This, mainly, depends on the combination of *Class* and *Object Property* query term nodes, which belong to the *visited query tree (sub)path*, as well as the contents of the corresponding SMS's. Thus inferred *equi–join* operations take also into account how a relationship between *agents*, in the natural language roles as *Subject* and *Object*, has been implemented. An *equi–join* 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** relationships between entity sets.

In both cases, additional *equi–join* based restrictions need to be considered for the WHERE clause. The conditional restriction is built upon the input provided by the

XML meta–data specification of the logical database schema, as briefly presented above.

For instance, given that the visited edge is $< CTN, CTN >$, with CTN being a *Class term node* and mappings to different names of relations, such as **PATIENTADMIT** and **PATIENTDIS**, due to the corresponding SMS's (query tree state (D) of figure 2), an *equi–join* 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** and **PATIENTDIS** are written into the FROM clause, with additional tokens standing for the variable names, whereas the restriction **PATIENTADMIT.PATIENT_ID = PATIENTDIS.PATIENT_ID** is added to the WHERE clause.

Similarly, given the combination $< CTN, RTN, CTN >$, with RTN being an *Object Property term node*, in a visited path having **PATIENTADMIT**, **PATIENTTHER** and **PATIENTTHER** as corresponding SMS's, only the **PATIENTADMIT**, **PATIENTTHER** are considered for the FROM clause. To this extent, only the *equi–join* 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's

$<: PATIENTTHER >$

as assigned to both the RTN and one of the connected CTN query term nodes. It is also possible to consider nested tables as well as operators or operations, which are assigned to particular query tree nodes.

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 high level query tree has been fully traversed. Following our example, the constructed query, which reflects the query tree state (D), as depicted in figure 2, would have taken the form

SELECT PATIENTTHER.THRMBDAT
FROM PATIENTADMIT VAR1, PATIENTTHER VAR2, PATIENTDIS VAR3
WHERE VAR1.PATIENT_ID = VAR3.PATIENT_ID **AND**
VAR1.PATIENT_ID = VAR2.PATIENT_ID **AND**
VAR2.THRMBLYS = '1' **AND**
VAR2.WHYNOTHR = '3'

The same SQL statement would have been generated, if words from a different natural language were assigned as values of the *name* slot of the query term nodes. To this extent, the transformation or generation logic for database query language statements remains unchanged. *However, the generated, for example, SQL statement would have been meaningless in terms of the real–world semantics, which*

are not part of the SQL or other database specific query language specification.

5 Conclusion

We presented a querying approach and language aiming at simplifying the query construction and transformation toward database specific query languages in terms of syntactic, semantic (NL–specific semantics) and pragmatic (real–world semantics) parsing of queries, especially for cross–lingual communities in conjunction with advanced application domain vocabularies such as scientific or technical ones.

The construction of the query is driven by reasoning services, which rely on an ontology description of the application domain semantics and, therefore, provide a real–world semantics driven interactive query construction mechanism. This is conceived as enabling moves among query states, which reflect a partly constructed or intended query according to some particular word order type as an NL–specific semantics template.

Currently, the reasoning services are targeted at constructing queries according to the word order type *Subject–[Verb–Object]*, which underlies almost 42 percent of the known natural languages such as English and German. However, the reasoning services can be slightly changed in order to guide the user to the construction of the *Subject–[Object–Verb]* word order type, which underlies almost another 45 percent of known natural languages such as Japanese and Turkish, without any change to the ontology description.

To this extent, the impact is twofold: (a) query generation by and specification of an automaton, where semantics guide the syntax rather than having the syntax guiding the semantics, (b) construction of a query regardless the preferred natural (sub)language, despite the diversity of word order types, with the same *extensional semantics* in terms of the tuples received with the query result.

The latter is guaranteed through the NL–independent reflection of the intended query on a high level query tree which, consequently, gets transformed into database specific query languages such as SQL. This is achieved by considering not only the nature of the submitted high level query tree, but also the mappings between ontology elements and storage model elements such as tables, attributes, etc., as well as a meta–data based description of the logical database schema(s).

The NL–independent reflection is also strengthened by the fact that mappings between ontology elements and storage model ones are established between *term unique identifiers* of the ontology elements and the storage model ones. This also alleviates the task of providing the same real–world semantics in more than one natural language by simply replacing the values for *names* of the ontology elements.

References

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

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

[3] B. Bouchou, D. Maurel, Natural Language Database Query System, in: 4th International Conference on Applications of Natural Language to Information Systems, IOS Press, Klagenfurt, Austria, 1999.

[4] R. H. Chiang, C. E. H. Cecil, V. 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, IOS Press, Versailles, France, 2000, pp. 215–226.

[5] A. Düsterhöft, B. Thalheim (Eds.), Natural Language Processing and Information Systems, 8th International Conference on Applications of Natural Language to Information Systems, LNI, GI, Burg (Spreewald), Germany, 2003.

[6] V. Ambriola, V. Gervasi, Experiences with Domain-Based Parsing of Natural Language Requirements, in: 4th International Conference on Applications of Natural Language to Information Systems, IOS Press, Klagenfurt, Austria, 1999.

[7] M. Mittendorfer, W. Winiwarter, Experiments with the Use of Syntactic Analysis in Information Retrieval, in: 6th International Conference on Applications of Natural Language to Information Systems, IOS Press, Madrid, Spain, 2001, pp. 37–44.

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

[9] S. C. Kleene, Automata studies, Princeton Univ. Press, Princeton, N.J., 1956, Ch. Representation of events in nerve nets and fi nite automata, pp. 3–42.

[10] D. E. Knuth, Semantics of Context-Free Languages, in: Mathematical Systems Theory, Vol. 2, 1968, pp. 127–145.

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

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

[13] H. Jaakkola, B. Thalheim, Visual SQL? High-Quality ER-Based Query Treatment, in: M. A. Jeusfeld, O. Pastor (Eds.), Conceptual Modeling for Novel Application Domains, ER 2003 Workshops, ECOMO, IWCMQ, AOIS, and XSDMER, Vol. 2814 of LNCS, Springer, Chicago, USA, 2003, pp. 129–139.

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

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

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

[17] S. Bechhofer, R. Stevens, G. Ng, A. Jacoby, C. Goble, Guiding the User: An Ontology Driven Interface, in: N. W. Paton, T. Griffi ths (Eds.), Proc. User Interfaces to Data Intensive Systems (UIDIS99), IEEE Press, Edinburgh, 1999, pp. 158–161.

[18] F. Baader, D. Calvanese, D. McGuinness, D. Nardi, P. Patel-Schneider (Eds.), The Description Logic Handbook: Theory, Implementation and Applications, Cambridge University Press, 2003.

[19] V. Sugumaran, V. C. Storey, Creating and Managing Domain Ontologies for Database Design, in: 6th International Conference on Applications of Natural Language to Information Systems, IOS Press, Madrid, Spain, 2001.

[20] V. Sugumaran, V. C. Storey, An Ontology Based Framework for Generating and Improving DB Design, in: 7th International Workshop on Applications of Natural Language to Information Systems, Springer Verlag, Stockholm, Sweden, 2002, pp. 1–12.

[21] K. Knight, S. Luck, Building a Large Knowledge Base for Machine Translation, in: Proc. of the AAAI-94, Seattle, USA, 1994, pp. 779–784.

[22] K. Knight, I. Chancer, M. Haines, V. Hatzivassiloglou, E.-H. Hovy, M. Iida, S. K. Luk, R. Whitney, K. Yamada, Filling Knowledge Gaps in a Broad-Coverage Machine Translation System, in: Proc. of IJCAI95, Montreal, Canada, 1995, pp. 1390–1397.

[23] B. Thalheim, T. Kobienia, Generating DB Queries for Web NL Requests Using Schema Information and DB Content, in: A. M. Moreno, R. P. van de Riet (Eds.), NLDB 2001, Vol. 3 of LNI, GI, Madrid, Spain, 2001, pp. 205–209.

[24] E. Metais, H. C. Mayr, NLDB'99: Applications of natural language to information systems, Journal of Data and Knowledge Engineering 35 (2) (2000) 107–109.

[25] M. C. Baker, The Atoms of Language, Oxford University Press, 2002.

[26] I. Horrocks, P. F. Patel-Schneider, F. van Harmelen, From SHIQ and RDF to OWL: The Making of a Web Ontology Language, Journal of Web Semantics 1 (1).

[27] N. F. Noy, M. Sintek, S. Decker, M. Crubezy, R. W. Fergerson, M. A. Musen, Creating Semantic Web Contents with Protege-2000, IEEE Intelligent Systems 16 (2) (2001) 60–71.