MDDQL: an ontology driven, multi-lingual query language and system for an integrated view of heterogeneous data sources.

Epaminondas Kapetanios
Panagiotis Chountas

Harrow School of Computer Science

This is an electronic version of a paper presented at the Semantic Technology Conference 2005, 7th-10th March 2005, California, USA.

MDDQL: An Ontology Driven, Multi-lingual Query Language and System for an Integrated View of Heterogeneous Data Sources

E. Kapetanios and P. Chountas

School of Computer Science
University of Westminster
Watford Road, Northwick Park, Harrow HA1 3TP
London, United Kingdom

E.Kapetanios@wmin.ac.uk

ABSTRACT

Query languages and keywords based search engines are conventionally specified and implemented with the emphasis put on syntactic rules to which query typing and answering must be bound. MDDQL is a query language and system that operates on a semantic model in terms of a graph based ontology. As a software technology, MDDQL allows the meaning of/and associations between information to be known and processed at execution time at following levels: (a) driving the user to the construction of, as meaningful as possible, queries with an advanced concept-based search method, (b) resolving high level queries into various data source specific query statements. In addition, queries can be posed in more than one natural sub-language. The major goal behind this approach has been the *simplification* and *scalability* of both tasks: query construction, even within multi-lingual user communities, and addressing of a large number of possibly semantically heterogeneous data sources in a distributed environment.

Keywords
Query Languages, Ontology, Concept-based Search, Semantic Data Integration, Conceptual Mediation, Semantic Technology

INTRODUCTION

During the last years, we are witnessing an increased awareness of and expectations from the emergence of semantic technologies as a promising field, where some answers and solutions to problems in intelligent search and data integration, natural language processing, knowledge representation and management. These expectations have also been stimulated by the vision of the Semantic Web [9, 10].

A concept based querying language and system, however, has been one of the objectives and visions behind this semantic wave as taking us one step beyond keyword based information retrieval techniques [15].

Riding on this semantic wave, MDDQL has been developed as a multilingual, concept based query language and system and applied in medical applications during the last 3-4 years. In addition, MDDQL has been extended towards a querying system for collections of, eventually, heterogeneous databases.

Despite the fact that ontology driven querying and intelligent data integration has been the focus of research activities and commercially available solutions in the last decades, the overwhelming amount of data available on the Web or even within a well defined business or organizational environment still poses some considerable challenges to be met.

Two of these challenges, *simplicity* and *scalability*, have been the major concern behind the MDDQL development, since we were aiming at:

- Turning MDDQL into a concept based querying system and paradigm. However, not only by using a kind of *taxonomies*, but also allowing within query expressions *semantic relationships or associations* other than classification hierarchies, operators, negation, etc., as already has been the case for many years in theory for database query languages.
- Cope with more than one natural language as a means of expressing a query.
- Turning MDDQL into a data integration system for ad-hoc integration of data sources, in particular, databases, however, avoiding customizing of interfaces [11, 12] between

middleware and data source, i.e., between mediators and wrappers

In order to meet these challenges (a) in terms of simplifying the query construction, resolution and result presentation process, (b) in terms of scalable ad-hoc integration of databases, despite the embedding of various semantic descriptions such as quality parameters, semantic distance, etc., a Semantic Engine with an Ontological model has been considered as the major part of the system.

Organization of the paper: Semantic Engine and Ontological model are described in the following section. The model, however, is realized in terms of a multi-layered graph, since different roles of terms (linguistics, ontology, perspectives, etc.) need to be addressed within the same ontology.

The following sections describe the impact of the Ontological model and Semantic Engine

- on to the ontology driven query construction mechanism as based on suggestions inferred by the system to refine the query,
- on to the query resolution and distribution mechanism at the mediation level in terms of delegating high level query trees to the data sources instead of *rewriting queries* as known by other data integration approaches,
- on to query result synthesis and presentation as performed in terms of M-Operators and in terms of wrapping results with all relevant metadata at the presentation level too.

SEMANTIC ENGINE AND SYSTEM ACHITECTURE

MDDQL qualifies as a semantic model driven query language and system, since its architecture incorporates a *Semantic Engine* (SE) as fueled by an Ontology [13, 14]. The purpose, however, of the SE is twofold: (a) it drives the human-computer interaction logic (IL) for the concept based construction of queries, (b) provides the mechanism for resolving the high level queries into various database specific queries and, subsequently, for the synthesis of partially received query results.

Despite the fact that this architecture, as depicted by **Error! Reference source not found.**, reflects a classical 3-tier architecture, with the SE simply replacing the term *Middleware* or *Mediation* within other architectural approaches, there are also some considerable differences to be taken into account. They mostly refer to the *Semantic Interfaces* (SI) and the underlying Ontological model.

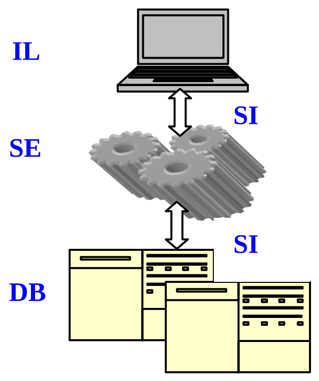

Figure 1: MDDQL Semantic Engine in a 3-tier System Architecture

The SI between SE and IL refers to those objects or abstract data types, which carry on

- those portions of the ontology to be transferred to or queried from a smart client machine or device,
- the submitted high level MDDQL query for resolution and distribution
- the synthesized final query result to be presented to the user.

The SI between SE and DB refers to the objects or abstract data types, which carry on

- those parts from the MDDQL high level query tree in form of a high level sub-query tree, which clearly refers to a particular database or, generalizing, to a particular data source
- the returned query results annotated through additional semantic descriptions such as origin, data quality, etc.

In the following, we will refer to the Ontological Model, which underlies the functionality of the *Semantic Engine* in terms of inferences as drawn for the needs of driving

- the user to the construction of concept based queries
- the query resolution and query result synthesis

The principles of the inferences and their added value to the whole system with respect to these two aspects are described in the following two sections. However, prior to embarking on describing the interaction and query resolution / query result synthesis logic, we give a general description of the Ontological Model, since

- this is strongly related with the kind of inferences we draw (graph traversing and navigation algorithms)
- it provides the vocabulary for the query language.

Ontological Model and Vocabulary: **The basics** The vocabulary of MDDQL is described by an ontology language, which is close to the Ontology Web Language [13] as proposed by the Web Ontology Working Group of the Wide Web Consortium (W3C) for the Semantic Web.

However, given that we put the emphasis on simplicity and scalability of maintenance and reasoning, the ontology model of the semantic engine relies on conceptual graph based formalism, which addresses the nodes and links as being objects themselves. This, in turn, makes persistent storage and maintenance straight forward given that it is closer to object-oriented paradigms of data models and management systems. Figure **2** depicts an example of such a graph based description.

This is similar to a surface syntax as based on *frames*. Frames group together information about each class or instance and, therefore, make an ontology easier to read and understand, particularly for those who are not familiar - or do not want to become familiar with *Description Logics* based formalisms [16]. The frames paradigm has been used in a number of well known knowledge representation systems such as OKBC [14].

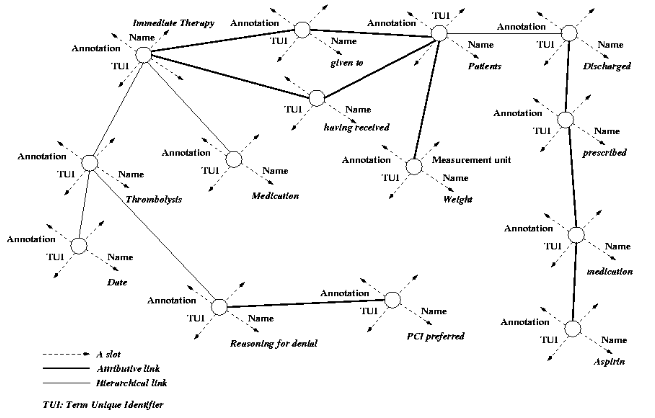

Figure 2: An Annotated Graph Based Ontology Representation

Roughly speaking, 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, specifying more general properties or having inverse property relationships.

Ontological Model and Vocabulary: **The differences** However, given that we use separate frames in order to represent terms for relationships, e.g., *received by, admitted to, etc.*, properties, e.g., *age,* and domain values, e.g., *[20-120]*, in order to describe concepts and instances, we reserve links for the descriptions of constraints and relationships. This holds not only among concepts and instances, but also among their descriptive elements such as *relationships*, *properties* and *domain values*.

To this extent, further semantic relationships or roles can be expressed upon all constituting elements of the ontology such as those known by natural languages, e.g., *subject, object* in the roles of *agents.* Links, on the other side, can be characterized as *attributive* or *hierarchical* links (see also Figure **2**) by assigning them specific slots, since they are also represented as objects with their own properties.

Generally speaking, this modeling and representation technique enables the addition of various semantic description layers for all elements of the graph or ontology, respectively. One such layer is the *constraints* layer which might add validity issues for holding or expressed connections among graph or ontology elements.

All these semantic layers refer to a particular perspective point of view such as roles within a natural language, roles within the ontology description, constraints, naming, etc., from which nodes within the conceptual graph should be observed.

This separation between *naming* of nodes and their roles within the graph or, respectively, ontology, alleviates the task of taking into consideration multi-lingual perspective points of view. It is, for instance, quite easy to change or add labels for naming of nodes in another natural language, while preserving all other semantic relationships and descriptions and with no need to adapt either the software for interaction logic or the software for the query resolution and transformation.

This is due to the fact that the conceptual graph is being mainly traversed according to the connectivity layer, where links among the nodes are established on the basis of *Term Unique Identifiers* (TUI's) as depicted by Figure **2**.

ONTOLOGY DRIVEN QUERY CONSTRUCTION

The first impact of the MDDQL *Semantic Engine* refers to the interaction logic underlying the query construction technique. It strongly relies on making suggestions of how to complete or construct a query through suggestions of conceptual relationships such as classification hierarchies, properties, value restrictions, operators, etc.

The set of suggested terms, however, depends on

- the semantic relationships as circumscribed by the Ontological Model (see also previous section)
- the partly constructed query as a whole,
- the role of the query term, from which a refinement has been triggered,
- the user community.

To this extent, the interaction logic resembles the moves among potential states as specified by a language automaton. It is this ontology driven automaton, which qualifies MDDQL as a concept based query language to be applied to domains with an advanced and hardly understood terminology such as scientific or technical ones, as well as a concept based search mechanism.

This concept based querying paradigm enables a better exploitation of the query vocabulary and results and, therefore, enables experimentation with ad-hoc queries within advanced application domains such as technical and scientific ones.

This is strengthened by the fact that, in contrast with keywords [15] or syntax only based querying paradigms, the user does not need to know the spelling of words (query terms) or their intentional meaning in order to construct a proper query.

Furthermore, given that queries are constructed by describing semantic relationships rather than simply typing keywords, the received query results are closer to the intentional meaning of the query.

Each constructed query at the client or smart device, however, is reflected and represented by a high level, conceptual tree, which is specified by constraints such as

- *the root of the query tree is always a Class or an Instance term node,*
- *an Object Property term node, i.e., a relationship between two agents, must have children, which are Classes or Instance term nodes*
- *an Object Property term node, i.e., relationship between two agents, might also have as children Property term nodes,*
- etc.

All query tree nodes, however, are reflections of the graph nodes as provided by the Ontological Model and, therefore, still carry on all those semantic layers, which refer to the various roles of these nodes (Figure 3). To this extent, it is easy to identify each query tree node either as a concept or class, property, instance, etc. Moreover, addressing the semantic layer of mappings to the data sources, as represented by the MID (Mediation Identifier) slot (Figure 3), it is easy to map each node to the corresponding data source elements.

Given also that naming or labeling of the nodes is devoted a separate semantic layer, submission of a query with query terms expressed in a different natural language would lead to the same query tree construction. Since transformation of the MDDQL query tree takes place by considering other semantic layers than the naming semantic layer, the same query results can be generated independently the preferred natural language, which underlies the expression of the terms.

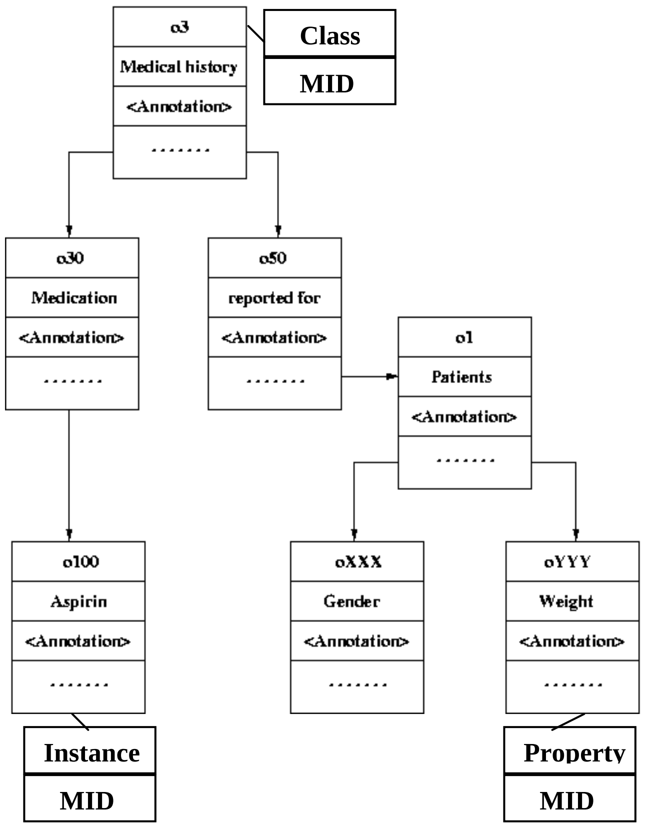

Figure 3: An Example of a Multi-Layered MDDQL Query Tree

In the following, we will describe how the query tree transformation takes place soon after its transmission as a query object to the semantic engine at the middle tier is completed.

SEMANTIC ENGINE AND QUERY RESOLUTION

The contribution of the MDDQL semantic engine to *Intelligent Data Integration* can be best described having first understood the major current approaches and philosophies behind querying of collections of databases or data sources.

Formally speaking, a data integration system **I** is defined by the triple **<G, S, M>** where **G** is the global schema expressed in a language **Lg** with an alphabet **Ag** , **S** is the source schema expressed in a language **Ls** with an alphabet **As**. In our case, G is given by the Ontological Model, where S is described by an RDF-like syntax in XML.

Global-As-View: Integration information from pre-selected sources according to a set of predefined information needs. A procedural approach is known (TSIMMIS, Squirrel, WHIPS) to integrate information from sources through ad-hoc procedures. When information needs or sources change, a new mediator should be generated.

The TSIMMIS [1] query language is a SQL like language adapted to deal with OEM objects. Adding new information to TSIMMIS requires building of a wrapper for the source and the change of all the mediators that will use the new source. It further has to be stressed that global integration is never performed in the context of TSIMMIS.

As a result a certain concept may be treated in different and possibly inconsistent ways, by different mediators. The TSIMMIS query converter supports queries that syntactically match a template and queries that produce the same results as a directly supported query. The notion of logical equivalence is used to detect queries that fall in this class. Queries that can be executed in two steps: first a directly supported query is executed, and then a filter is applied to the results of the first step.

Squirrel [2], WHIPS [3] share the goal of providing a query-based approach to data amalgamation. However maintenance of views against updates to the sources is the main aspect in this context. The focus here is on the timeliness and availability of data.

Local-As-View: Integration information from arbitrary sources according to a set of predefined information needs. A declarative approach is known (Information Manifold, Carnot, SIMS,). Mediators contain mechanisms to rewrite queries according to source descriptions. A rewritten query should be contained in the original query.

In the Information Manifold [4] a reasoning phase is delivered for realizing which sources have the data of interest, unlike TSIMMIS where view expansion is used for finding what data each source must contribute. To resolve queries, a mapping between the relations in the mediated schema and the source relations is defined. A method to define these mappings is to describe each source relation as the result of a conjunctive query, over the relations in the mediated schema. The collection of available data sources may not contain all the information needed to answer a query.

In SIMS [5] sources are described using a domain model of the application domain that it is formalized in terms of a class based representation language (LOOM). Query processing involves a non-fixed mapping from query to sources that are dynamically selected and integrated when the query is submitted.

In Carnot [6] system individual schemata are mapped onto a large ontology, which is provided by the CYC knowledge base. Such ontology is expressed in an extended first order representation language called Global Context Language (GCL). Source schemata and global views are represented in a knowledge base, and an inference engine based on rules is used to extract information from the sources.

Given that **M** is the mapping between **G** and **S** in some kind of description language **Lm**, this is either based on definition of *views* or on other partially defined Ontologies. In any case, query resolution and delegation to data sources becomes a matter of *query rewriting*, which increases complexity. In addition, this mapping description approach decreases scalability when it comes to ad-hoc integration of data sources.

In summary the problem of query answering in *Mediators* can be formally described as follows: given a set of views $v_1 \dots v_k$ and a query Q over a fixed schema, can Q be reformulated using the views so that it does not use any of the base relations? The problem is in principle NP-complete with respect to set semantics.

View suitability is parameterized by languages in which views and queries are expressed and by the semantics under which they are evaluated. The focus is on two special cases known as the equivalence problem (do two queries Q_1, Q_2 return the same set of answers?) and the containment problem (is the set of answers to Q_1 always a subset of those to Q_2 ?).

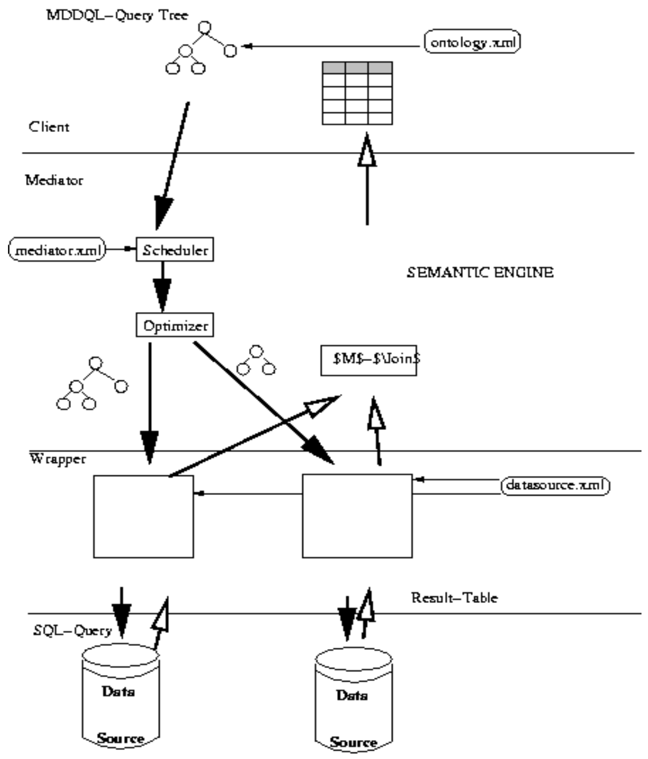

Figure 4: Overview of Query Resolution in MDDQL

MDDQL approach: In order to simplify query resolution and answering over integrated databases, we rely on *simplicity* and, therefore, *scalability* of ad-hoc data source integration. This is enabled by taking into consideration:

- The semantic descriptions of the nodes on the MDDQL query tree (Figure 3), in particular, their Ontological roles such as *Concepts, Properties, Relationships, Instances*, etc.

- The descriptions of mappings from Ontology-to-Data source elements, where data source elements are described in terms of *containment paths (SMS)*.

An SMS or containment path is defined by the constraint that all SMS constituents underlie a sequence order like

<data source>:<table>:<attribute>:<value>

as for databases, a notation which indicates inclusion, i.e., an **<attribute>** is included by a **<table>**, a <**table**> is included by a **<data source>.** Note that **<table>,<attribute>,<value>** might also refer to recursive structures such as nested tables, complex attributes, and multi-values.

Similarly, in cases of documents, the SMS structure follows a sequence order like

<data source>:<document>:<paragraph>:<text>

An example of these mapping descriptions in terms of resolving MID's, as expressed by additional layers of the nodes on the MDDQL query tree (Figure 3) into SMS's and for the database world is given in the following:

```
<mddql:mediator>
  <mddql:mid mid="m100">
    <mddql:sms distance="1.0">
      AMIS:PATIENTADMIT
    </mddql:sms>
    <mddql:sms distance="1.0">
      CCT2003:ANGIO_PATIENTS
    </mddql:sms>
    <mddql:sms distance="1.0">
      CCT2003:REVA_PATIENTS
    </mddql:sms>
  </mddql:mid>
  <mddql:mid mid="m501">
    <mddql:sms distance="1.0">
      AMIS:PATIENTADMIT:SEX
    </mddql:sms>
    <mddql:sms distance="1.0">
      CCT2003:ANGIO_PATIENTS:GENDER
    </mddql:sms>
    <mddql:sms distance="1.0">
      CCT2003:REVA_PATIENTS:GENDER
    </mddql:sms>
  </mddql:mid>
  . . . . . .
</mddql:mediator>
```

This way of structuring mappings also enables the consideration of additional elements such as semantic distance. The major goal behind this structure, however, is easiness of ad-hoc extensions through new data sources, since new data source elements in terms of containment paths should be simply added as an XML element into the relevant parent XML element as it refers to the Ontology concept through the MID attribute.

This mapping notation is also exploited by the query resolution and distribution algorithm, since the source or origin of data is a-priori known. To this extent, the MDDQL query tree can be split up into sub-query trees, however, each sub-query tree includes only those nodes, which refer to containment paths of the same origin. Consequently, each sub-query tree can be delegated for transformation and execution to a particular database or data source.

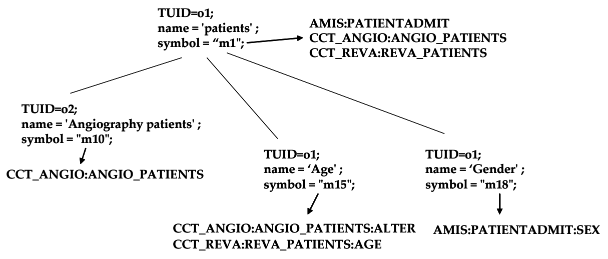

Figure 5: Example of MDDQL Query Tree Resolution

An example of an MDDQL query tree is depicted by Figure 5. The query tree refers to the query "*Age and gender of patients with angiography*". Accordingly, there are more than three (3) databases affected, as indicated by the resolved containment paths, namely, AMIS, CCT_ANGIO and CCT_REVA.

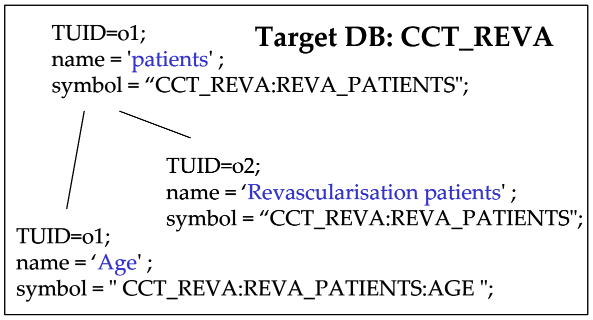
Target DB: CCT_REVA
TUID=o1;
name = 'patients' ;
symbol = "CCT_REVA:REVA_PATIENTS";
TUID=o2;
name = 'Revascularisation patients' ;
symbol = "CCT_REVA:REVA_PATIENTS";
TUID=o1;
name = 'Age' ;
symbol = " CCT_REVA:REVA_PATIENTS:AGE ";

Figure 6: An Example of a Generated Data Source Specific Query Tree

Following the example of Figure 5, Figure 6 depicts one of the three (3) query sub-trees as generated for the data source CCT_REVA.

Since each high level query sub-tree is clearly assigned to a particular data source, it is sent for transformation and execution to that particular source (Figure **4**). However, in order to accomplish its transformation towards a data source or database specific query language, e.g., SQL statements, three aspects need to be taken into consideration as input by the transformation algorithm:

- the nature of the high level query sub-tree in terms of pairs of nodes and their roles within the Ontology as the query tree is being traversed in a *depth-first* strategy
- the nature of the containment paths on each node

- the metadata description of the data source as provided by wrappers in an RDF-like syntax

These metadata as provided by the wrapper refers to, e.g., database schema description in terms of tables, attributes, primary/foreign keys, data types, measurement units, etc., as well as to quality parameters, e.g., completeness and soundness.

SYNTHESIS OF THE QUERY RESULT AND ITS PRESENTATION

Given that the partial query is executed at a dedicated source, all relevant metadata are propagated with the generated query result back to the mediator. The partial query results as returned from each source are merged at the mediation level through the application of the mediation operators **M**-Join, **M**-Union and **M**-Difference.

This merging addresses issues like putting attributes together or shifting values in cases of overlapping of attributes from different data sources. A major assumption is that all partially created query results are returned with their ID's, which are implicitly added to each sub-query result, if not explicitly requested by the query. Given that ID's are also subject to mapping descriptions, these **M**-operations are applied after having transformed ID's back to their Ontological mappings.

The integrated query result is presented to the user as a wrapped result in terms of

- references to the partially generated query results and the generated data source specific queries
- references to data source specific metadata such as quality at various granularity levels (source, table, attribute)
- references to the Ontological counterparts for explanations of attributes, tables, etc.

CONCLUSION

MDDQL has been conceived as a multi-lingual, concept-based querying language and system. It aims at going beyond *keywords based Information Retrieval*. However, as a *Concept-based Querying* language does not rely only on the *usage of Taxonomies* but also relationships, operators, operations, etc.

Given that its functionality and current application is driven by multi-lingual user communities and different Ontology perspectives and views as well as by *integration of* various collections of heterogeneous data sources, the emphasis has been put on *simplicity* and *scalability* to cope with the inherited complexity.

This refers to both the Ontological model as represented by a multi-layered conceptual graph and the way of describing Ontology-to-Data source mappings in terms of containment paths. The latter decreases complexity when ad-hoc data sources are being integrated.

MDDQL has been implemented in Java with its successful application for querying a series of databases as they refer to collections of patients' records as provided by a case study for the proof of technology.

Currently the system is extended towards querying and integration of unstructured data, e.g., Web documents in conjunction with the elaboration of the Ontological Model to accommodate different perspectives and views and other relativity issues within an Ontology.

Some questions still to be answered: Is there any contribution to the Semantic Web community or could this become an intelligent, concept-based querying Search Engine? Does it make sense to turn the query interface to a purely natural language based one?

ACKNOWLEDGEMENTS

We would like to express our thanks to Paul Groenewoud, David Baer, Markus Zingg, Elly Zarakani at Semantic Technologies GmbH, who co-implemented the system.

REFERENCES

1. J. Ullman, "*Information Integration using local views*", ICDT, pp. 19-40, LNCS 1186, Springer, 1997
2. G. Zhou, R. Hull, R. King "*Generating data integration mediators that use materialization*", Journal of Intelligent Information Systems, 6(2), pp. 199-221, 1996
3. J. Hammer, H. Garcia-Molina, J. Widom, W Labio, Y. Zhuge "*The Stanford data warehousing project*", IEEE Data Eng. Bulletin, Spec. Issue on Materialized Views and Data Warehousing, 18(2), pp. 41-48, 1995
4. A. Levy, A. Rajaraman, J. Ordille "*Query answering algorithms for information agents*" In Proc. of 13th National Conference of Artificial Intelligence *AAAI*, pp. 40-47, AAAI/MIT Press, 1996
5. Y., Arens, C. Knoblock, W., Chen "*Query reformulation for dynamic information integration*", Journal of Intelligent Information Systems, 6(2), pp. 99-130, 1996
6. M. Huhns, N. Jacobs, T. Ksiezyk, W. Shen, M. Singh, P. Cannata "*Integrating enterprise information models in Carnot*", CoopIS, pp. 32-42, IEEE Computet Society 1993
7. P. Mitra,"*An Algorithm for Answering Queries Efficiently Using Views*", in Proc. of ADC, pp. 99-106, IEEE Computet Society, 2001

8. R. Pottinger A. Levy, “*A scalable algorithm for answering queries using views*” pp 484-495, VLDB, 2000

9. T. Berners Lee, *Weaving the Web*, London, Orion Business, 1999

10. D. Fensel, J, Hendler, H. Liebermann (editors), *Spinning the Semantic Web*, MIT Press, 2003

11. G. Wiederhold, Mediators in the Architecture of Future Information Systems, IEEE Computer, 25(3), pp. 38-49, 1992

12. Y. Papakonstantinou, A. Gupta, H. Garcia-Molina, J.D. Ullman, *A Query Translation Scheme for Rapid Implementation of Wrappers*, Proc. 4th Annual Conf. on Deductive and Object-Oriented Databases (DOOD), pp. 161-186, Singapore, 1995, Springer

13. http://www.w3.org/2001/sw/WebOnt

14. http://ontolingua.stanford.edu/okbc

15. http://www.google.com

16. F. Baader, D. Calvanese, D. McGuiness, D. Nardi, P.-P. Schneider, *The Descriptions Logic Handbook: Theory, Implementation and Applications,* Cambridge Press, 2003