Diploma Thesis

A Context-Based Transformation
of High-Level Conceptual Query Trees

David Baer
email: dbaer@acm.org
Supervisor: Dr. Epaminondas Kapetanios

Prof. Dr. Gustavo Alonso

Information and Communication Systems Group
Institute for Pervasive Computing
Dept. of Computer Science
Swiss Federal Institute of Technology (ETH)
ETH-Zentrum, CH-8092 Zurich, Switzerland

January 11, 2004
Abstract

Creating a unified view on structured data residing in different, heterogeneous data sources is a problem that exists in many application domains. Various approaches have been proposed to address it. In this thesis, a different approach is taken by using an existing, ontology based querying tool and extending it to cope with heterogeneous data sources. The approach uses a mapping from ontology symbols to data source symbols on a high-level query tree. A RDF (Resource Description Framework) compliant description of data sources is proposed. Furthermore, a prototype implementation is presented. The approach is compared with other, description logic based, data integration approaches.
## Contents

1 Introduction
   1.1 Glossary .......................................................... 4
   1.2 Structure .......................................................... 5

2 Related Work ....................................................... 6

3 System Architecture ............................................. 8
   3.1 Comparison with Description Logic Approaches ............... 8

4 Resource Descriptions ........................................ 12
   4.1 Mediator-Level: Ontology-to-Database Mapping ............... 12
      4.1.1 Semantic Distance ........................................ 14
   4.2 Wrapper Level: Database Structure ............................ 15
      4.2.1 A Proposal for an RDF-Compliant Database Description . 16
      4.2.2 wrapper.xml Example ........................................ 16
      4.2.3 Quality Aspects ............................................ 19
   4.3 Creation of Meta Data Information ........................... 23
   4.4 Unit Conversion ................................................... 24
   4.5 Natural Language Translation .................................. 24

5 Algorithms .......................................................... 25
   5.1 Mediator ........................................................... 25
      5.1.1 Downwards: Mapping, Cloning and Splitting Trees ....... 25
      5.1.2 Upwards: M-Join ............................................. 29
      5.1.3 Other approaches ............................................ 30
   5.2 Wrapper ............................................................ 35

6 Implementation .................................................... 37
   6.1 Programming Environment ...................................... 37
   6.2 Data Sources ..................................................... 37
   6.3 Meta Data Creator ................................................. 39
   6.4 MDDQmulti ........................................................ 39
      6.4.1 Benchmarking ............................................... 44
   6.5 Open Issues ....................................................... 45

7 Conclusion .......................................................... 47
   7.1 Future Work ....................................................... 47

8 Acknowledgments .................................................. 49
<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>References</td>
<td>50</td>
</tr>
<tr>
<td>List of Figures</td>
<td>54</td>
</tr>
<tr>
<td>List of Tables</td>
<td>55</td>
</tr>
</tbody>
</table>
1 Introduction

In a very real sense, data constitutes the raw material for the Information Age [42]. The value of access to trustworthy data is of high importance to decision makers. Big companies are merging, divisions within companies are being reorganized on a repeated basis and valuable data sources are shared (with or without cost) within research institutions. As a result it is essential to be able to integrate and compare various data sources, without losing quality aspects of the original data. The challenge of integrating different data sources is faced by many different users of structured data, as much in economics as in science or in governmental administrations [36].

The focus of this thesis is the integration of heterogeneous, well structured data hosted in different data sources for a user asking queries to these data sources. It is restricted to queries (i.e. it is read-only) and gives the user the impression of dealing with a single data source. An existing framework (MDDQL\textsuperscript{1}) of an ontology-driven, graphical query interface [21] needed to be extended and adapted to enable the handling of queries to heterogenous data sources. The main technique used to dissolve heterogeneities is mapping the ontology onto the storage medium elements.

To achieve these goals, several aspects have been involved:

- The overall system architecture had to be designed. This was done using a standard client-mediator-wrapper-data source model.

- The data sources had to be described in a systematic way. In this thesis, a RDF\textsuperscript{2}-compliant description of data sources is proposed.

- The distribution of the queries to the different data sources and the following integration of the results needed algorithms. Different algorithms have been evaluated.

- The resulting data is enriched with information about the origin and the quality. Quality measures have been introduced.

- A prototype environment has been implemented in Java to extend the existing MDDQL-system. This prototype version is called MDDQLMulti (section 6.4). In addition, a Java tool supporting administrators in creating the data source description was realized as a prototype.

\textsuperscript{1}Meaning Driven Data Query Language

\textsuperscript{2}Resource Description Framework, see section 4.2.1 for a brief introduction.
1.1 Glossary

Throughout this document, the term high-level query tree is used to distinguish from SQL or relational algebra query trees used to resolve queries. A synonymous expression to “high-level query tree” sometimes used in the literature is conceptual query tree.

Ontology is the branch of metaphysics that studies the nature of existence [5]. In terms of knowledge systems and artificial intelligence, an ontology is the result of a study of categories of things that exist in a specific domain, which is a catalog of types that are assumed to exist in that domain. The ontology consists of the vocabulary and rules of a language used to discuss topics in the domain described. Within the MDDQL-system, queries can be asked about properties (columns in the result) which are assigned to entities (entities can be related by relations). A value is either used as a restriction within a query or represents a single cell in the result. This information is stored in the ontology. The result of a query within MDDQL is a high-level query tree.

A MDDQL high-level query tree is created by an Inference Engine (IE). Its structure is as follows: The root of the tree is always an Entity Term Node (ETN). The child of an ETN is either another ETN, a Relationship Term Node (RTN) or a Property Term Node (PTN). A RTN links between two ETNs. A PTN can either be a leaf, a parent of another PTN or of a Value Term Node (VTN) [21], [20].

When referring to the stored data, the term data source is used. This emphasizes the fact, that a data source could be represented in any structured form. The most common case however will be a (relational) database. When relating to databases, generally the terms table and relation are used synonymously, as well as the terms column and attribute. The other aspects within tables are rows (tuples), consisting of values. A tuple is identified uniquely from other tuples by the key or identifier.

Within this paper, the abbreviations MID and SMS are widely used. MID stands for Mediator Identifier and uniquely identifies concepts from the ontology. SMS is short for Storage Medium Symbol and represents uniquely databases, tables, columns and values. The format of these identifiers will be introduced in the appropriate chapters.

The query answering system discussed in this thesis is distributed and integrates several entities working together (see figure 1): There is the client side, which offers the user a graphical query interface and presents the result. The client interacts with the mediator. A mediator “mediates” between several different entities, i.e. it encapsulates the entities and defines how they interact [14]. In this case it mediates between the client and the data source wrapper and takes care of additional tasks as the fusion of the results. The wrapper (also known as adapter)
abstracts from the actual data sources.

Data integration researchers divide between two approaches defined and described in [25] as follows:

**GAV** The *global-as-view* approach requires that the global schema is expressed in terms of the data sources. Adding new data sources to a GAV-system is difficult – it may require a redesign of the global schema. Query processing on the other hand is simple.

**LAV** In the *local-as-view* approach, the global schema is created independently from the sources. Adding a new source simply means enriching the mapping with a new assertion, without other changes. The difficulties start with query processing, when only partial information about the data is available in the global schema.

A third way called *GLAV* combines some aspects of the two approaches mentioned above [13].

### 1.2 Structure

This paper is organized as follows. The next section consists of a brief overview of *related work* that in one way or another influenced the outcome of this document. It intends also to describe the vicinity of this work. It is succeeded by an introduction into the *system architecture* (section 3). Section 4 presents both the mapping- and the resource *description files* needed alongside some related topics. In section 5 the necessary *algorithms* are described. Section 6 provides a summary of *implementation* issues. It describes the structure and the programming environment as well as presenting some results of a few benchmark tests. Finally, section 7 concludes the thesis by summarizing insights of the thesis and by mentioning some open problems.
2 Related Work

This thesis aims to integrate the extensive work of many different aspects in the field of heterogeneous data sources into an existing application. [21] gives a good overview of the existing system MDDQL. Using strictly an ontology based approach in providing a user friendly front end to a well structured data source. It could until now deal only with homogeneous data sources. The approach followed in this thesis considerably simplifies the way of describing relationships between an ontology and heterogeneous database elements.

A good analysis of the problems encountered in this work is [9], considering data quality aspects as well as data provenance and the interplay of meta data and ontologies in the semantic integration of data. Nice categorizations of data quality are provided in [40] and [34], the latter also describing assessment practices. [38] provides an overview of possible source descriptors, including quality measures and additional information about data sources.

Another topic of this thesis is semantic heterogeneity. A measurement for describing semantic heterogeneities (semantic distance) used in this thesis is described in [8]. An interesting classification of conflicts that may arise is provided in [24].

Donald Kossmann describes in [22] many optimization strategies in query processing. So far, optimization considerations have not yet been introduced into the system. A relatively old study on what was then called “Federated Databases” is provided in [37]. A completely different approach than the one presented here is “InfoSleuth” [4] which uses an agent-based concept for semantic information integration. Yet another approach is presented in [2], where a distribute and iterative system is presented which collects meta data.

[17] gives a very good overview of many problems considered in this thesis, particularly in relation to the integration of data from various databases and the use of a “read-only” mediator. It does not propose any concrete solutions though. The tutorial [25] and the presentation [12] give an excellent introduction into the field of data integration from a description logic perspective.

Another key aspect is the semantic web. A solution for semi-automatic semantic interoperability among data sources has been proposed in [1]. In the context of the semantic web, the Resource Description Framework (RDF) [23] and its “sister” OWL (Web Ontology Language) [39], which are both in the process of being proposed by the World Wide Web Consortium (W3C), are important keywords. Both are a model to describe information. While OWL operates more on the level of ontologies and is therefore beyond the scope of this work, RDF describes data.
resources as in chapter 4.
3 System Architecture

The system can be divided into four different levels (see figure 1). The work for this thesis has been restricted to the two “sandwiched” levels named mediator and wrapper. These two levels also represent logical layers.

The mediator receives as an input the MDDQL high-level query tree. With help of an “ontology to source description mapping file” (mediator.xml) it adds data source-specific information to the high-level query tree. It is, therefore, at the logical layer of (data source) links and distributes the high-level query tree to different data sources. Also, nodes not relevant to the respective data source are removed from the tree. As a result, the mediator forwards high-level sub query trees enriched with the real data source symbol (SMS) to the wrappers.

The wrapper represents the logical layer of data source schema description and may or may not be on the same physical machine as the data source or the mediator. The meta data file wrapper.xml contains all necessary information to build a SQL query from the MDDQL high-level query tree received from the mediator. Upon receiving the result from the data source, the wrapper transforms it into the types and the structure desired by the user. This process ensures that the mediator will receive homogeneous data.

The mediator “joins” (using $M \Join -$ — the mediator-join) the results together to one single result table. This table is then forwarded back to the user.

These two layers also differ in the way they deal with different heterogeneities [41]: the mediator emphasizes structural heterogeneity (schematic heterogeneity) by creating different high-level (sub) query trees with the according data source symbols. The wrapper takes care of semantic heterogeneity (data heterogeneity) in the way that it transforms the results into a common format.

In the following chapters, these layers will be considered and described in a more detailed way under several aspects.

The two other layers are not part of this thesis. The client level only needs minor adaptations. The data source level does not need to be changed at all. In fact, one of the goals is to be able to plug in new data sources without having to adjust them.

3.1 Comparison with Description Logic Approaches

The design described above is standard for many proposals in data source integration [10], [15], [17] and especially [22]. From a description logic point of view, it cannot be classified properly as a GAV (global-as-view) or a LAV (local-as-view) approach. It is in between the two approaches and in fact seems to combine the advantages of both of them.

First, a very brief overview of the notation used in [25] by Lenzerini is given. It
Figure 1: The “big picture” of the system architecture.
is followed by describing how the MDDQL-approach fits in. “The main com-
ponents of every data integration system are the global schema, the sources, and the
mapping. This is formalized as a data integration system \( I \) in terms of a triple
\( (\mathcal{G}, \mathcal{S}, \mathcal{M}) \), where

- \( \mathcal{G} \) is the global schema, expressed in a language \( \mathcal{L}_G \) over an alphabet \( \mathcal{A}_G \).
The alphabet comprises a symbol for each element of \( \mathcal{G} \) (i.e., relation if \( \mathcal{G} \)
is relational, class if \( \mathcal{G} \) is object-oriented, etc.).

- \( \mathcal{S} \) is the source schema, expressed in a language \( \mathcal{L}_S \) over an alphabet \( \mathcal{A}_S \).
The alphabet \( \mathcal{A}_S \) includes a symbol for each element of the sources.

- \( \mathcal{M} \) is the mapping between \( \mathcal{G} \) and \( \mathcal{S} \), constituted by a set of assertions of the forms

\[
q_S \leftrightarrow q_G,
q_G \leftrightarrow q_S
\]

where \( q_S \) and \( q_G \) are two queries of the same arity, respectively over the
source schema \( \mathcal{S} \), and over the global schema \( \mathcal{G} \). (…)

Intuitively, the source schema describes the structure of the sources, where the
real data are, while the global schema provides a reconciled, integrated, and virtual
view of the underlying sources. The assertions in the mapping establish the
connection between the elements of the global schema and those of the source
schema.

Queries to \( I \) are posed in terms of the global schema \( \mathcal{G} \), and are expressed in
a query language \( \mathcal{L}_G \) over the alphabet \( \mathcal{A}_G \). A query is intended to provide the
specification of which data to extract from the virtual database represented by the
integration system.”

The MDDQL-approach is clearly an integration system \( I \) as described by Lenz-
erini:

- The global schema \( \mathcal{G} \) is the ontology, defining its own language \( \mathcal{L}_G \) and
alphabet \( \mathcal{A}_G \) specified in [20] and [21].

- The source schema \( \mathcal{S} \) is represented by the wrapper description file pre-

- The mapping \( \mathcal{M} \) between \( \mathcal{G} \) and \( \mathcal{S} \) is represented through the mediator meta
data file described in section 4.1. A difference is, however, that in the MD-
DQL approach, the mapping is not a mapping of queries over the respective
alphabets, but rather a mapping of symbols of the alphabets directly. For-

mally expressed, the mapping \( \mathcal{M} \) in MDDQL is constituted by a set of assertions of the forms
where \( a_S \) and \( a_G \) are symbols representing elements of \( S \) or \( G \) respectively. Intuitively, an assertion \( a_G \rightsquigarrow a_S \) specifies that the concept represented by the symbol \( a_G \) in the global schema \( G \) is expressed in a source schema \( S \) by the symbol \( a_S \) (similarly for an assertion of type \( a_S \rightsquigarrow a_G \), which is represented only implicitly).

It seems that the difference in the definition of the mapping has extensive implications:

- The approach falls in between the definitions of LAV and GAV. On the one hand, modeling and adding a new data source is in general as simple as in the LAV approach. It simply needs to extend the mapping \( M \). On the other hand, the MDDQL approach does not have the complications of the LAV approach when it comes to queries, because the queries are generated at the sources. In other words, when a query \( q \) is posed over the alphabet \( A_G \), it does not substitute every element of \( A_G \) with the corresponding query over the sources as in the classical GAV approach. The MDDQL approach does only substitute (or more precisely add) symbols of respective source elements. The query construction still takes place locally at the source using the source specific alphabet \( A_S \).

- Through these differences, the expressiveness and/or the flexibility of MDDQL might suffer. Furthermore, it is not clear at this point how studies on complexity and soundness/completeness of queries in a LA-/GA-view are affected. Indeed, these questions need further investigation.
4 Resource Descriptions

This section focuses on various aspects concerning meta data. Each of the two levels of the system described in this thesis (see chapter 3) need their own description of the data sources and the inter-connections (of course, the ontology has a description file too, but this is not part of this thesis – and so has the database its internal meta data). Accordingly, the two description files will be commented separately in the next paragraphs.

Due to its wide acceptance the eXtensible Markup Language (XML) [6] has been chosen as data format for both files. Furthermore, the file describing the data source implements the Resource Description Framework (RDF) [23] for which a new terminology describing databases has been implemented.

4.1 Mediator-Level: Ontology-to-Database Mapping

The mediator is responsible for mapping the MID (Mediator Identifier) to the respective Storage Medium Symbol (SMS). The relation between MID and source symbol is $n : m$, meaning that an MID-element may contain several source symbols and a source symbol could show up at several different MID-elements. A MID-element mapping into several different data sources means, that this ontology concept is represented several times within the data sources, possibly in different sources. Also it gives information to the mediator about attributes in different sources that can be united or joined. It is among these attributes that the mediator will infer M-joins. This is only possible, because the data source-wrapper guarantees that these attributes will be transformed into equivalent values (same unit, same scale, same data type or even explicit transformations for keys, for example).

The source symbol elements contain an attribute named distance (distance), which is described in paragraph 4.1.1. The homogeneous key-element (mddql:homogeneousKey) is needed for the fusion of the partial results obtained from the different data sources. A join can be accomplished over any two key-elements (mddql:key) within one homogeneous key-element. The schema definition 1 shows the XML-Schema [11] representation of the mediator description file structure.

Schema-Definition 1

```xml
<?xml version="1.0"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">
  <xs:element name="mddql:mediator" type="mediatorInfo"/>
  <xs:element name="mddql:homogeneousKeys" type="homKeyInfo"/>
  <xs:complexType name="mediatorInfo">
    <xs:element name="mddql:mid" type="midInfo">
      <xs:attribute name="mid" type="xs:string" use="required"/>
    </xs:element>
    <xs:element name="mddql:homogeneousKeys" type="homKeyInfo"/>
  </xs:complexType>
</xs:schema>
```
A short example is presented below (see example 1). There are two different data sources: AMIS and CCT2003. Both of them have a table with an attribute (“SEX” in “AMIS”, “GENDER” in “CCT2003”) which is identical to the ontology property (term node) Gender (mid="m501"). Element mid="m1" represents a mapping from an Entity Term Node onto a table within the data sources. The concept (“Patients”) exists in both data sources. In “AMIS”, the respective table is called PATIENTADMIT. In the “CCT2003” source it exists in two different tables (ANGIO_PATIENTS and REVA_PATIENTS). Gender contains only the two categorical values “F” (mid="m1000") and “M” (mid="m1001"). In this example there is one key which is inter-operable (and therefore can be used for joins) within the two data sources: the three mddql:key-elements that are descendents of the mddql:homogeneousKey-element.

The storage medium symbol is defined in a similar way as it is already known from the MDDQL-concept [21]. The general representation for a value is: database: relation:attribute:value, for an attribute (column) it is database:relation:attribute respectively and so forth.

Example 1

<mediator>
  <mid mid="m1">
    <sms distance="1.0">
      AMIS:PATIENTADMIT
    </sms>
    <sms distance="1.0">
      CCT2003:ANGIO_PATIENTS
    </sms>
  </mediator>
4.1.1 Semantic Distance

The system description file metadata.xml contains one single quality attribute, the semantic distance coefficient $d$. The concept is widely described in the literature. According to [35], it indicates how much the concept expressed in the ontology $O$ is represented by the data source $S$. $d$ belongs to the real interval $[0, 1]$. In other words, $d$ characterizes the capability of the concept associated with $O$ to characterize the concept associated with $S$. The semantic distance is located on the level of information heterogeneity within information systems and from a broader view of social interaction it is at the semantic-level according to the classifications collected in [33]. The interval for possible values is from 1.0 (meaning semantically absolutely
equivalent) to 0.0 (representing that there is no semantic equivalence at all). Because of the case where \( d = 0.0 \) implies that there is no semantic relation at all between the two concepts, and this is best represented by not mentioning it, the effective interval in mediator.xml has been restricted to \((0, 1]\) (0-exclusive).

4.2 Wrapper Level: Database Structure

Two main tasks have to be accomplished by the wrapper: It first transforms the MDDQL high-level sub query tree into a database specific SQL-query. Upon execution it transforms the results into the desired (by the query) formats. For both tasks, the wrapper needs to access the database meta data. For the first task, we can fall back on the existing homogeneous MDDQL-system.
Only minor modifications are needed. The wrapper needs information about the database schema, the data types and the references within the table. To transform the results into a common format as required by the query, the wrapper needs information about the scale, the unit and the transformation table. The quality measures are not used for the query at this stage, but will be added to the result, leaving it to the mediator and the user to decide how to use them.

### 4.2.1 A Proposal for an RDF-Compliant Database Description

The need for meta data about resources available on the web is critical for the semantic web. The World Wide Web Consortium (W3C) is therefore in the process of defining such a framework called Resource Description Framework (RDF) [23]. So far, there is no standard (known to the author) for describing databases yet. For this reason, a RDF-compliant proposal is given here. Figure 2 shows a graphical representation of the proposed description. It contains only a few of the quality measures described in section 4.2.3.

The basic idea is that a database consists of tables and references between the tables. A table is built by several attributes. Each attribute has certain properties, such as data type, natural language, unit and scale. In addition, information on possible transformations is given. There can be either a one-to-one mapping between values, the values can be mapped into a range or the name of a complete class providing complicated transformation mechanisms can be indicated. A special measure should be taken to provide exchange rates, possibly by indicating a link to a resource keeping current (and if needed older) exchange rates. A unit conversion table is not part of the resource description (see section 4.4). A reference is built from one or more couples of keys.

A similar description has been proposed in [16]. However, it is a simple description, which ignores quality aspects, transformation tables, owner and other information not contained in the original database.

Figure 2 shows a graphical RDFS representation [7] of the proposed description. It contains only a few of the quality measures described in section 4.2.3 in definition 1.

### 4.2.2 wrapper.xml Example

Example 2 (below) shows a sample data source description. The database AMIS contains two tables (attributes in brackets): `PATIENTADMIT` (PATIENT_ID, HOSPREC_ID, SEX), `CONDITION` (PATIENT_ID, HOSPREC_ID, SUBMISSION_DATE). The attributes PATIENT_ID and HOSPREC_ID from PATIENTADMIT reference the respective attributes in CONDITION. The values of the
attribute PATIENTADMIT.SEX in AMIS can be female or male, while the ontology allows only the values f or m. Therefore they need to be transformed.

This example contains only a few of the quality aspects presented in definition 1.

Example 2
<?xml version="1.0" encoding="ISO-8859-1"?>
  xmlns:db="http://pliro.org/2003/10/13-databaseDescriptionTerms"
  xmlns:q="http://pliro.org/2003/10/13-databaseQualityTerms"
  xmlns:dc="http://purl.org/dc/elements/1.1/"
  xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
  xml:base="http://pliro.org/mediTerms">
  <db:database rdf:ID="AMIS">
    <db:name login="LUZR" pw="GUESS" url="iks3.inf.ethz.ch:1521:oraIKS3"
      propertiesPath="/home/david/DBProps.xml">
      AMIS
    </db:name>
    <db:vendor-version>Oracle/9</db:vendor-version>
    <db:schema>LUZR</db:schema>
    <db:references rdf:parseType="Collection">
      <db:crossReference db:cardinality="" rdf:parseTape="Collection">
        <db:referenceElement>
          <db:refElement1>AMIS:PATIENTADMIT:PATIENT_ID</db:refElement1>
          <db:refElement2>AMIS:CONDITION:PATIENT_ID</db:refElement2>
        </db:referenceElement>
        <db:referenceElement>
          <db:refElement1>AMIS:PATIENTADMIT:HOSPREC_ID</db:refElement1>
          <db:refElement2>AMIS:CONDITION:HOSPREC_ID</db:refElement2>
        </db:referenceElement>
      </db:crossReference>
    </db:references>
  <dc:creator>Daniel Gygax</dc:creator>
  <db:owner>Alain Nef</db:owner>
  <db:creationDate rdf:datatype="&amp;xsl:date"/>
  <q:completeness rdf:datatype="&amp;xsl:decimal">1.0</q:completeness>
  <q:soundness rdf:datatype="&amp;xsl:decimal">0.9</q:soundness>
  <db:schema rdf:parseType="Collection">
    <db:name>PATIENTADMIT</db:name>
    <db:table rdf:ID="AMIS:PATIENTADMIT">
      <q:completeness rdf:datatype="&amp;xsl:decimal">1.0</q:completeness>
      <q:soundness rdf:datatype="&amp;xsl:decimal">1.0</q:soundness>
      <db:attributes rdf:parseType="Collection">
        <db:attribute rdf:ID="AMIS:PATIENTADMIT:PATIENT_ID">
          <db:name>PATIENT_ID</db:name>
          <q:completeness rdf:datatype="&amp;xsl:decimal">1.0</q:completeness>
          <q:soundness rdf:datatype="&amp;xsl:decimal">1.0</q:soundness>
          <db:key>primary</db:key>
        </db:attribute>
      </db:attributes>
    </db:table>
  </db:schema>
</db:database>
</rdf:RDF>
<db:attribute rdf:ID="AMIS:PATIENTADMIT:HOSPREC_ID">
    <db:name>HOSPREC_ID</db:name>
    <q:completeness rdf:datatype="&amp;xsl;decimal"/>1.0</q:completeness>
    <q:soundness rdf:datatype="&amp;xsl;decimal"/>1.0</q:soundness>
    <db:key>primary</db:key>
    <db:type>VARCHAR2</db:type>
    <db:language>english</db:language>
</db:attribute>

<db:attribute rdf:ID="AMIS:PATIENTADMIT:SEX">
    <db:name>SEX</db:name>
    <q:completeness rdf:datatype="&amp;xsl;decimal"/>1.0</q:completeness>
    <q:soundness rdf:datatype="&amp;xsl;decimal"/>1.0</q:soundness>
    <db:key>none</db:key>
    <db:type>VARCHAR2</db:type>
    <db:language>english</db:language>
</db:attribute>

<db:transformationTable rdf:parseType="Collection">
    <db:transformElement db:transformMapping="F" db:transformValue="female"/>
    <db:transformElement db:transformMapping="M" db:transformValue="male"/>
</db:transformationTable>

<db:attribute rdf:ID="AMIS:CONDITION:PATIENT_ID">
    <db:name>PATIENT_ID</db:name>
    <q:completeness rdf:datatype="&amp;xsl;decimal"/>1.0</q:completeness>
</db:attribute>

<db:table rdf:ID="AMIS:CONDITION">
    <db:name>CONDITION</db:name>
    <q:completeness rdf:datatype="&amp;xsl;decimal"/>1.0</q:completeness>
</db:table>
4.2.3 Quality Aspects

Quality has been defined as fitness for use [18]. There is a lot of interesting work in the field of information quality (see [3], [34], [40] and [43]).

The wrapper level has different quality aspects in comparison to the mediator layer. Here information about the quality of the data is considered, regardless of the ontology. The work of [19] is used to narrow the term “quality”. The authors use a table (see table 1) to differentiate several aspects of information quality in what they call the *PSP/IQ model* (product and service performance model for information quality).

<table>
<thead>
<tr>
<th>Attribute ID</th>
<th>Attribute Name</th>
<th>Completeness</th>
<th>Soundness</th>
</tr>
</thead>
<tbody>
<tr>
<td>AMIS:CONDITION:HOSPREC_ID</td>
<td>HOSPREC_ID</td>
<td>1.0</td>
<td>1.0</td>
</tr>
<tr>
<td>AMIS:CONDITION:SUBMISSION_DATE</td>
<td>SUBMISSION_DATE</td>
<td>1.0</td>
<td>1.0</td>
</tr>
</tbody>
</table>

19
Conforms to Specifications | Meets or Exceeds Consumer Expectations

<table>
<thead>
<tr>
<th>Product Quality</th>
<th>Sound Information</th>
<th>Useful Information</th>
</tr>
</thead>
<tbody>
<tr>
<td>The characteristics of the information supplied meet IQ standards</td>
<td>The information supplied meets information consumer task needs.</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Service Quality</th>
<th>Dependable Information</th>
<th>Usable Information</th>
</tr>
</thead>
<tbody>
<tr>
<td>The process of converting data into information meets standards.</td>
<td>The process of converting data into information exceeds information consumer needs.</td>
<td></td>
</tr>
</tbody>
</table>

Table 1: Aspects of the PSP/IQ model in [19]

For the purpose of this thesis (specify the quality of a database, table or attribute), mainly the “Sound Information” quadrant is interesting. To decide whether the information retrieved is useful or usable (see respective quadrants) must be decided by users individually. It is highly dependent on the task. Some of its dimensions (Relevancy, Understandability, Value-Added) are of some interest, because they meet with the goal for the complete MDDQL-system. The dimensions of the “Dependable Information” quadrant have to be met by the database administrators and the system itself. Dependable information is current, secure and provided in a timely manner to support the task at hand. It cannot be evaluated a priori from characteristics of data in a database. This becomes more obvious when we see how information quality (IQ) dimensions are mapped into the PSP/IQ model [19] (see table 2).

The importance and poor standard of the sound-dimensions was shown in a study presented in [19]: The IQ of sound-dimensions was below average and undercut only by the usable-dimensions. In [40], data quality (DQ) dimensions are categorized into four different categories (Intrinsic DQ, Accessibility DQ, Contextual DQ, Representational DQ).

The IQ dimensions in table 2 and relevant to this thesis have been defined as follows:

**Completeness** is the extent to which information is not missing and is of sufficient breadth and depth for the task at hand.

**Concise Representation** is the extent to which information is compactly presented.

**Consistent Representation** is the extent to which information is presented in the same format.
<table>
<thead>
<tr>
<th>Product Quality</th>
<th>Sound Information</th>
<th>Useful Information</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>• Free-of-Error</td>
<td>• Appropriate Amount</td>
</tr>
<tr>
<td></td>
<td>• Concise Representation</td>
<td>• Relevancy</td>
</tr>
<tr>
<td></td>
<td>• Completeness</td>
<td>• Understandability</td>
</tr>
<tr>
<td></td>
<td>• Consistent Representation</td>
<td>• Interpretability</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Service Quality</th>
<th>Dependable Information</th>
<th>Usable Information</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>• Timeliness</td>
<td>• Believability</td>
</tr>
<tr>
<td></td>
<td>• Security</td>
<td>• Accessibility</td>
</tr>
</tbody>
</table>

Table 2: Mapping the IQ dimensions into the PSP/IQ model as in [19]

**Free-of-Error** is the extent to which information is correct and reliable. The term “free-of-error” is sometimes also referred to as soundness.

**Relevancy** is the extent to which information is applicable and helpful for the task at hand.

**Understandability** is the extent to which information is easily comprehended.

**Value-Added** is the extent to which information is beneficial and provides advantages from its use.

**Timeliness** is the extent to which the information is sufficiently up-to-date for the task at hand.

Additional dimensions for the purpose of molecular biology information systems have been proposed in [30]:

**Reliability** is the accuracy of experimental method with which the data is produced.

**Price** is the monetary price of a query in US-Dollars.

In [27] a more formal description of soundness and completeness is given. A database and the real world which it models are formalized as two instances of the same database scheme. The actual (stored) database instance is denoted by $D$, while the ideal (real-world) database instance is denoted by $W$. Of course,
$W$ is a hypothetical instance which is unavailable. The stored instance $D$ is an *approximation* of the ideal instance $W$. The following estimations of soundness and completeness should be based on samples of $D$ and $W$ because they may be very large.

- **Soundness** (of the data source relative to the real world: $\frac{|D \cap W|}{|D|}$)
- **Completeness** (of the data source relative to the real world: $\frac{|D \cap W|}{|W|}$)

Furthermore a practical technique on how to estimate these values has been described:

**Estimating Soundness:** To determine the proportion of the sorted information that is true, the following procedure is used:

1. Sample $D$.
2. For each $x \in D_{sample}$ determine if $x \in W$.
3. Calculate the soundness estimate as the number of “hits” divided by the sample size $|D_{sample}|$.

Step 1, sampling the database, is simple. But Step 2 is difficult: we need to determine the presence of database elements in $W$ without actually constructing it. This is accomplished by *human verification* of the sample elements.

**Estimating Completeness:** To determine the proportion of the true information that is stored, the following procedure is used:

1. Sample $W$.
2. For each $x \in W_{sample}$ determine if $x \in D$.
3. Calculate the completeness estimate as the number of “hits” divided by the sample size $|W_{sample}|$.

Step 2, verifying the presence of elements in the database, is simple. But Step 1 is difficult: we need to sample $W$ without actually constructing it. This may be interpreted as constructing a *fair challenge* to the database by using various resources (e.g., by judicious sampling of alternative databases).

The following quality dimensions have been considered important and necessary for the purpose of sufficiently informing the user about the quality of the information retrieved from various data sources:
Definition 1  **Quality Dimensions:**

**Completeness** as defined above: the proportion of the stored information that is true in the database compared with the real world.

**Soundness** as defined above: the proportion of true information that is stored in the database compared with the real world.

**Timeliness** is defined by the currency of a data source. It is easily computable, but depends on the application. Exchange rates may need high refreshment rates, while measurements of physical experiments may tolerate relatively old data.

*Other dimensions can easily be introduced according to the application.*

All these quality dimensions are rated on the real interval $[0, 1]$. A value close to 0 indicates that the quality dimension is not at all met while a 1 indicates that the data complies perfectly with the quality dimension. The quality dimensions are based on the three levels data source ($qd_{ds}$), tables ($qd_{tbl}$) and attributes ($qd_{att}$). The quality dimension of a retrieved value is then the product of these three values.

**Definition 2**  **Quality Dimension of a retrieved value:**

$$qd_{val} = qd_{ds} \times qd_{tbl} \times qd_{att}.$$  

4.3 Creation of Meta Data Information

Another problem is the creation of meta data. Although this is not the main topic of this thesis, in order to implement a working prototype system it also needs to be considered.

For the wrapper-description file a basic tool assisting with a graphical user interface has been implemented in Java (section 6.3). It retrieves as much data known to the database as possible (as table- and attribute names, key and type information) and offers interfaces to easily create other information that is often incomplete (references) or not available (quality measures, unit, scale etc.). To create transformations it selects all values currently in the database to manually transform them into the values desired by the ontology (or the user).

As for the prototype implementation, the meta data needed by the mediator has been edited by hand. It is desirable to create at least a tool that assists in this process.
4.4 Unit Conversion

The process for the creation of a standard for the computerized description of units and its interdependencies has not been accomplished yet. Unix operating systems include a program called units [26], which uses the /usr/share/units.dat file. The cml-project (chemical markup language) [28] contains a nice descriptions of units in XML [29]. An overview of the related problems and the current state of affairs in building a universal unit description can be found in [31] and [32].

4.5 Natural Language Translation

A translation service for natural languages is provided by the ontology server. It should already contain many translations, so it can offer a translation service without a huge additional effort.
5 Algorithms

For the mediator, the existing MDDQL framework had to be extended with newly implemented algorithms. On the wrapper-level the algorithms generating the SQL-query needed minor adaptations compared to the previous version for homogeneous data sources. Everything concerning transformation of values is newly added.

5.1 Mediator

The algorithms to be performed on the mediator-level can be roughly divided into two categories:

- On the “way down” from the client to the data source, the mediator must map the identifier assigned by the ontology (the MID) to the storage medium symbol (SMS). A MID can map onto one or more SMS’ on at least one data source. The high-level query trees passed on to the wrapper, however, should only contain nodes with exactly one SMS. Therefore, the mediator must clone trees and prune them in order to provide the data source wrappers with data source specific high-level query trees.

- After retrieving the results from the various data sources, the mediator must join the different tables together. This represents the “way up” on figure 1.

5.1.1 Downwards: Mapping, Cloning and Splitting Trees

The problem of mapping the MID’s onto the SMS is straightforward. It is solved with the help of the mediator.xml meta data file (see section 4.1).

The high-level query tree now contains nodes where the MID maps onto more than one SMS (on different sources). For each source, a new tree will be cloned, containing only the mappings that belong to that source. These cloned trees may also contain nodes that contain no SMS, because the concept does not exist in this source. These “empty” nodes are removed from the high-level query tree by pruning the branches.

Even if heterogeneity appears within the same data source (i.e. a concept from the ontology maps onto several different SMS within the same data source), the system can cope with this by treating it as virtually different data sources. See example 3 for a better understanding.

Example 3 The transformations at the mediator upon receiving a high-level query tree

A query constructed on the client side by the MDDQL query construction tool (see
is shown in figure 3. The mediator receives the respective query tree shown in figure 4.

The first thing that must be done on the mediator is to map the MIDs on the SMS’. 

Figure 3: A query in the MDDQL-querying tool. This query asks for the dates of birth and the height as well as the symptoms of all female patients that have been admitted to the hospitals with a recorded condition.

While doing this, several cases are possible:

- **The simplest case**: all nodes in the high-level query tree belong to the same data source and there is only one mapping per node. The high-level query tree can be sent directly to that data source.

- **More than one data source is involved, but they do not overlap**. This case is represented in figure 5. The edge, which has one node in one data source and the other node in an other data source represents a M-join. To both of the two data sources Shibuya and Shinshuku, a sub tree is sent containing only the nodes which have a representation in that data source. Still we are considering a case, where there is exactly one mapping per node.

---

3These different cases can obviously not represent a real data source. Therefore the virtual names Shibuya and Shinshuku are used instead of AMIS or CCT
Figure 4: A high-level query tree. In the high-level query tree that results from the query in figure 3, $E$ represents Entities, $R$ represents a Relation-node, $P$ stands for Property and $V$ symbolizes Value term nodes as described in [21].

Figure 5: The high-level query tree for several data sources without intersection after splitting.
The data sources are overlapping. This only has an influence if Property Term Nodes are involved. At this stage, this case does not influence the mediator’s algorithm. Upon retrieving the results from the different data sources, the mediator must enforce a union over these attributes. The mediator can still generate separate high-level query (sub) trees and submit them to the appropriate data source as presented in figure 6.

The most difficult case occurs when a MID maps on several SMS of the same data source. This is a manifestation of a data source having two attributes describing the same objects. It is expected that this case will not happen regularly as this is poor database design, but the system can still master such a situation by splitting the tree and sending the sub trees separately.

Figure 6: The high-level query tree for different overlapping data sources. This implies a union (U).
to the same data source. It handles this case as two (or more) completely separate data sources and therefore generates separate queries (see figure 7). The resulting tables will be joined and unified by the mediator.

Figure 7: Several MIDs are mapped on more than one SMS within the same data source

5.1.2 Upwards: M-Join

After sending the completed high-level sub query trees to the wrappers, the mediator awaits the results from all wrappers. The mediator has then to merge these results to one single result that can be sent back to the client. As shown in figures 6 and 7, this may lead to joins and unions. As it is not always very clear whether a union or a join or both is needed, the \textit{M-Join} (\(M - \Join\)) is introduced. By first implementing a \textit{full outer join} (Algorithm 1) over the two tables and then \textit{shift-}
ing” (Algorithm 2) equivalent columns together, the problem of choosing a join or a union can be omitted, the M-Join does it both.

Algorithm 1: M-Join Part 1: Fusion

Input: All tables returned by the wrappers $T$

Output: A single table $t_{temp}$ containing all tuples

Algorithm 1 implements a simple full outer join over all tables returned by the wrapper. It joins the largest common subset of keys available in the temporary table with the keys from the table that is currently being joined. The information on which of the columns are keys and which are equivalent is stored in the mediator description file mediator.xml (see section 4.1). The resulting table has as many columns as the sum of the number of columns of all tables involved. Various strategies for best implementing these joins can be chosen [22].

Figure 8 shows a possible M-Join over three tables $A$, $B$ and $C$. In the first step a full outer join of table $A$ and table $B$ over the keys $A.\alpha$ and $B.\alpha$ is invoked. Tuples with equal keys result in the same row in the resulting table. The result is then joined with table $C$ over the pair of keys $A.\alpha = B.\alpha = C.\alpha$ and $B.\beta = C.\gamma$. In the second step (algorithm 2) the resulting table is shifted together, removing columns not desired to be seen by the user and “fusioning” equivalent columns. If two equivalent columns contain different values in the same row, the one with better quality will be chosen. The other values (with inferior quality) will be forwarded to the user too, so the user may decide which results she/he wants to use. Different values on equivalent values should not happen. If they occur, it is because of “wrong” values in one of the involved data sources.

5.1.3 Other approaches

Definition 3 Let $G(V, E)$ be a connected graph with a finite set of vertices $V$ and a set of undirected edges $E$ with an associated weight function $w(e) \rightarrow (0, 1]$, $e \in E$. The problem MaxMul is defined as the problem of finding a spanning tree $MaxMulSpannTree T^*$ in $G$ that maximizes $\prod_{e \in T^*} w(e)$.
Leaf Property Term Nodes (PTN) from high level query tree:

Mappings: Column → PTN

Equivalent keys:

\[ A.\alpha \equiv B.\alpha \equiv C.\alpha \]
\[ B.\beta \equiv C.\gamma \]

Figure 8: A M-Join of 3 tables step-by-step. The three columns chosen by the user are the three leaf Property Term Nodes: 1 \((A.\alpha \equiv B.\alpha \equiv C.\alpha)\), 2 \((A.a \equiv B.a \equiv C.b)\) and 3 \((B.b \equiv C.c)\).
Algorithm 2: M-∞ Part 2: Shift

**Input**: A table $t$. The output of algorithm 1

**Output**: A table $t_{temp}$ containing every column only once

```plaintext
for all rows $r_i \in t$ do
    $r_i(t_{temp}) \leftarrow merge\&remove(r_i(t))$
end for
return $t_{temp}$
```

**merge\&remove**:

**Input**: A row $r$ from a table $t$ after algorithm 1

**Output**: A row $r_{final}$ where fields that are presented in the same column are merged and fields not to be presented are removed

$C_{presented} \leftarrow$ all columns in $t$

$C_{presented} \leftarrow$ all columns in $t$ to be presented to the user

```plaintext
for all $c_i \in C_{presented}$ do
    $C_{temp} \leftarrow r \cap c_i$
    $r_{final}[i] \leftarrow MAXQuality(C_{temp})$
end for
return $r_{final}$
```
Theorem 1  \(\text{MaxMulSpannTree} \leq_p \text{MinSpanningTree} \) (see definition 3) for a graph \(H(V,E) \) \( (V(H) = V(G), E(H) = E(G)) \) with a weight function \(w' = -\log(w(e)) \forall e \in E\). Note that \(w'(e) \geq 0 \forall e \in E \) because of \(w(e) \in (0,1]\).

**Proof:** Replace the function \(w(e)\) with \(w'(e)\) in \(G\). The correctness of the reduction is proved in theorem 2. \(\square\)

Theorem 2  To prove that the reduction in theorem 1 is correct, we prove that \(T^*\) is \(\text{MinSpanningTree}\) in \(H\) iff \(T^*\) satisfies \(\text{MaxMul}\) in \(G\).

**Proof:** \(T\) is any \(\text{SpanningTree}\), \(T^*\) is \(\text{MinSpanningTree}\) in \(H\), i.e.

\[
T^* := \text{MinSpanningTree in } H \iff \\
\sum_{e \in T^*} w'(e) \leq \sum_{f \in T} w'(f) \iff \\
\sum_{e \in T^*} -\log(w(e)) \leq \sum_{f \in T} -\log(w(f)) \iff \\
-\sum_{e \in T^*} \log(w(e)) \leq -\sum_{f \in T} \log(w(f)) \iff \\
\sum_{e \in T^*} \log(w(e)) \geq \sum_{f \in T} \log(w(f)) \iff \\
\log \left( \prod_{e \in T^*} (w(e)) \right) \geq \log \left( \prod_{f \in T} (w(f)) \right) \iff \\
\prod_{e \in T^*} (w(e)) \geq \prod_{f \in T} (w(f)) \iff \\
T^* \text{ is } \text{MaxMulSpannTree} \text{ in } G \quad \square
\]

The problem of finding the relevant data sources can be demonstrated graphically as a graph \((G(V,E))\), where nodes \((V)\) represent data sources and edges \((E)\) represent references of two attributes between two data sources (see figure 9). The weight of an edge \((w(e))\) is the semantic distance between the two attributes which are closest (in semantic meaning) to each other (see section 4.1.1).

A query involves a non-empty subset \((V_{involved})\) of all data sources. The task of finding all necessary data sources to be able to join the results of each data source together to one result is equivalent to finding a spanning tree over all involved nodes \(V_{involved}\) in the graph \(G\). The problem of finding the best of such spanning trees is (according to theorems 1 and 2) identical to finding a minimum spanning tree. Figure 10 shows an example with a query to three different data sources \((2, 6, 7)\). Data source 3 is only involved to provide the best possible join between sources 2 and 7 and source 6.
Figure 9: References between data sources. A graphical representation of various data sources and the references between these data sources. The weight of an edge represents the semantic distance between the two best (closest) attributes.

Figure 10: “Shortest path” to connect data sources. In this example, data sources 2, 6 and 7 are directly involved in the query. The respective spanning tree with maximum weight needs data source 3.
This approach tries to realize the joins between results over (possibly) data sources and/or tables originally not involved in the query. It would need to request other possible data sources and might come up with extremely complicated joins. Within the MDDQL-system however, it is guaranteed that all sub queries will generate results that can be joined directly. It is therefore not necessary and much safer in terms of scalability and response time to use this property and join the results directly.

5.2 Wrapper

The wrapper generates the data source specific SQL queries. This can be done with minor adaptations to the currently used MDDQL-algorithm described in [21]. The following modifications have to be introduced:

- **Retrieve identifiers for all Entity- and Relationship (ETN/RTN) nodes.** Because a high-level query tree can now represent subtrees of the original tree and because it is not known to the subtree, where other subtrees have been removed it is possible that the tables represented by these tables in the data source will be used to join different partial results from (different) data sources. Therefore, all ETN and RTN nodes visited during the query construction by the wrapper will have the key identifiers of the respective tables added to the query projection. They will not be shown to the end user (if not explicitly requested), but used on the mediator level to join (and unite) tables.

- **Entity- and Relation Term Nodes are permitted as leaves.** Until now, it was not permitted to have Entity- or Relation Term Nodes (ETN, RTN) as leaves of the high-level query tree. As a complete query, a tree containing ETN or RTN leaves would not make any sense. It represents a query about an abstract entity (or even relation between two entities) without specifying any interest in a property of that entity or using it as a restriction. Now, a high-level query tree sent to a wrapper can be part of a query or it may just be used to join partial queries. Therefore, it now makes perfect sense to allow ETN and RTN leaves. They will be treated as other ETN/RTN nodes and just retrieve all identification information from the respective tables as described above.

- **Value transformation.** Values have to be transformed from MDDQL-values into data source values and vice versa. The first transformation occurs before sending queries to the data source. Value restrictions in SQL “where-clauses” need to be mapped into the format specific to the respective data source. The description file `wrapper.xml`
(see section 4.2) defines these transformations. Also unit and currency conversions and scale transformations (e.g. from kilos to megas) need to be accomplished for restricting values. 

After submitting the query and receiving the result from the data source, resulting values have to be transformed back into the MDDQL-specific values, using again the transformation tables from wrapper.xml (inversed of course). Scale, currency and unit conversions need to be taken into account. The results are *enriched* with quality- and other information saved in the wrapper.xml-file.
6 Implementation

Part of this thesis is the implementation of a working prototype. MDDQL already consists of an application for homogeneous data sources. While the client side only needed minor modifications, the mediator had to be programmed completely new. The data source wrapper could re-use some of its code, mainly the translation from the high-level query tree to a SQL statement.

6.1 Programming Environment

As all code available so far was written in Java, the question of the implementation programming language never arose. Database access was provided by the Oracle-JDBC bridge. The parsing of XML-files was done using the Xerces Java parser.

6.2 Data Sources

The two underlying databases are both running on Oracle 9. The sources have the following characteristics:

**AMIS**
- The database contains about 2000 tuples per table for all schemas (users). The largest schema size is about 450 tuples.
- The design for “AMIS” is object-relational, containing about 120 attributes.

**CCT**
- ANGIO is a case study of necessity and appropriateness of angiographies. REVA is a case study of necessity and appropriateness of revascularizations (open heart surgery) at the Cardiocentro Ticino (CCT) in 2002.
- ANGIO contains roughly 2500 tuples per table, REVA about 1400.
- ANGIO and REVA both follow a relational design. ANGIO contains about 30 attributes, REVA about 20.

Unfortunately, the two sources do not have many fields in common. Therefore they are adequate for the purpose of showing the capabilities of MDDQLmulti only to a limited extent.
Figure 11: Meta data creator: Add attribute
6.3 Meta Data Creator

The Meta Data Creator is a simple tool which assists in generating the data source description file “wrapper.xml” (see sections 4.2 and 4.3). It is a simple graphical prototype programmed in Java. To write the XML-file, the Xerces parser is used, access to the database is provided by JDBC. It reads in the complete structure of the database. Figure 11 shows how the properties of an attribute can be entered and/or changed.

The tool assists in creating a transformation table by showing all (“DISTINCT”) values from the attribute. The user then can add the mapping as known to the ontology. Figure 12 shows an example for the attribute GENDER.

To create references between two tables, the tool presents all tables. Upon choosing two different tables, the user is presented a list of all available attributes in both tables. The user can easily build pairs of equivalent keys out of this list (see figure 13).

6.4 MDDQLmulti

The purpose of the implementation is to present a working example of MDDQL in a heterogeneous data source environment. The resulting application so far works properly. However, very limited effort has been given to performance issues. All operations have been implemented purely in the high-level programming language Java. The most difficult and time consuming operation is of course the join operation. The only performance enhancing measure taken was the use of a hash table.
Figure 13: Meta data creator: Create references
on the keys. But still, there is a lot of space left for improvement.

Figure 14 shows an overview of the packages in MDDQL. They are divided by

![Packages overview diagram]

Figure 14: Packages overview

the physical device (client, mediator, wrapper) they are running on. Following is

a brief description of each package:

**gui** The gui-package contains all classes used for the graphical query interface. This package was not touched by this thesis and is not maintained by the author of this thesis.

**clientCommunication** In this package, there are very few classes responsible for the communication between client and mediator. No adaptations were needed for this thesis.

**presentation** The presentation-package contains the classes needed for the result presentations. So far, no modifications were made in this package for the purpose of this thesis. However, the representations of quality and origin in the result (see section 6.5) needs adaptions in this package.

**mediator** All classes needed by the mediator are in this package. This package was created new from scratch for purpose of this thesis.

**wrapper** In the wrapper-package are mainly the classes needed for communication between mediator or database and the wrapper. The classes used for communication with the database did not need any changes. The other classes are new.
**wrapper.interpretation** The interpretation package consists of all classes needed for the transformation of the MDDQL high-level query tree to the SQL query. In this package, some minor adaptions were necessary to “upgrade” MDDQL from homogeneous to heterogeneous data sources.

**clientMedShare** This package is made up by classes shared between client and mediator. It contains only classes needed for communication between entities and did not need modifications for this thesis.

**medWrapShare** The classes in this namespace are shared between the mediator and the wrapper. They are used for the communication between the two parts. These classes are new.

**shareAll** In this package are the classes needed by the client, the mediator and the wrapper. It is built mainly by classes that are used for result-representations. It also contains helper classes needed by all three entities, mainly for the handling of SMS. Some adaptions were made in this package.

**ie** The ie-package contains the *inference engine* used for building and accessing the MDDQL high-level query tree [21]. It is not under responsibility of the author. Some minor modifications have been implemented for this thesis in the ie-package by Dr. Epaminondas Kapetanios who is the author of the classes in this package.

![Figure 15: UML-Diagram: mediator](image-url)
Figure 15 shows a UML-representation of major classes used by the mediator. The main class is `MediatorThread`. It is started whenever a new request reaches the mediator. The `SymbolResolver` implements the algorithms described in section 5.1.1. Through the class `MediatorXML` it accesses the `mediator.xml`-file (see section 4.1).

The M-Join (section 5.1.2) is implemented in the `Scheduler`. All values in the result are instances of the abstract super class `CellContent`.

Communication with the client is done through a sub class of `ClientConnection`, while `WrapperConnection` and its sub classes communicates with the wrapper.

The UML-diagram in figure 16 shows the main classes of the wrapper (section 5.2 describes the algorithms). Here too, a `WrapperThread` is started for each new incoming query. It hands on the `QueryTreeContainer` (formed of sub classes of `QueryTermNodes`) received to the `QueryGenerator` which builds a `QueryContainer`. Out of this “QueryContainer,” the `QueryCreator` then constructs the SQL-query string. Through the appropriate instance (depending on the database product) of a `DatabaseConnection`, this SQL query is sent to the database. The `WrapperXML`-class provides access to the `wrapper.xml`-file described in section 4.2.

The “WrapperThread” takes over the result from the “DatabaseConnection”. It transforms it into a table of `CellContent` (figure 15) instances and sends it back through the `MediatorConnection`.

Figure 16: UML-Diagram: wrapper
6.4.1 Benchmarking

Several benchmark tests have been conducted. This proved to be very difficult and of limited significance. The reason for the difficulties are:

- The MDDQL system, that requires graphical query creation, complicates systematic and specific querying.

- The test databases are limited in size. It is not possible, to create queries that produce answers of sufficient size to cause thrashing in memory. It is therefore not possible so far to define at which level the querying gets too slow.

- There is only one possible attribute to create joins. This attribute is of a “String” data type.

![Figure 17: Time in milliseconds for M-Joins](image)

Figure 17 shows a plot of several test measuring the time for the M-Join in milliseconds. For the above mentioned reasons, it is not very meaningful. At least, it
does not show that the time rises dramatically. The tests were conducted on a system described in table 3. The time used to join the data was never significant. The Internet connection (ADSL, 300 Kbit/s downstream, 50 Kbit/s upstream) always made up for about 20 to 30 seconds (compared to the less than 90 milliseconds for the M-Join). The shift-operations (see algorithm 2) was responsible for about 10 to 35 percent of the whole M-join operation, depending on the size of each tuple, while the fusion operation (algorithm 1) was more affected by additional tuples.

The tests also showed only a slight increase in time when joining three tables compared to join two tables of the same total size. Much more benchmarking is necessary and different versions of joining should be tested.

### 6.5 Open Issues

The following topics have not been implemented yet or are not good enough:

- As mentioned in section 6.4, a lot of improvement for performance is possible throughout the source code, especially in the join- and the communication parts.

- More testing is necessary to detect hidden bugs and to verify the correct working of the newly implemented code.

- The transformations between values so far can only handle transformation tables. Transformations between natural languages, currencies, units, ranges of values and classes as described in sections 4.2 to 4.5 is not implemented yet.

- The prototype version is running locally (having client, mediator and wrapper running on the same physical machine). While the system is designed that it can easily be adapted to run distributed, this is so far not implemented.

<table>
<thead>
<tr>
<th>Machine</th>
<th>IBM Think Pad T40</th>
</tr>
</thead>
<tbody>
<tr>
<td>Operating System</td>
<td>Debian Linux</td>
</tr>
<tr>
<td>Kernel</td>
<td>2.4.20-686</td>
</tr>
<tr>
<td>J2RE</td>
<td>1.4.2_01b06</td>
</tr>
<tr>
<td>Processor</td>
<td>Intel Pentium M 1500 MHz</td>
</tr>
<tr>
<td>RAM</td>
<td>512 MB</td>
</tr>
</tbody>
</table>

Table 3: Test system configuration.
• The exception handling is not sophisticated enough and needs some serious re-thinking.

• The system so far does not check access right. This issue needs some theoretical work.

• The result presentation is not yet capable of presenting additional information (quality, origin) the result already contains.
7 Conclusion

This thesis developed a theoretical framework for extending the MDDQL-system to cope with heterogenous data sources. Two different meta data files were needed to achieve this. A working prototype was implemented and evaluated. While the available resources showed to be of only limited use (because the overlapping attributes were too few), the prototype is running properly. Still, there is a lot of work left to transform this prototype into a “real world”-application, but major problems are solved.

The comparison with research results from GAV- and LAV approaches in section 3.1 showed that the approach presented here is very promising and seems to solve major problems in query construction or the creation of global schemas in a heterogeneous environment.

7.1 Future Work

Open issues in the program code are mentioned in section 6.5. Apart from this, it would be desirable to have a RDF-compliant description of the mediator meta data (described in section 4.1). The database description file presented in section 4.2 should be revised and might be used as an input for an international standardization process in creating a RDF-database description.

Benchmarking tests should be conducted on large scale basis with different data sources. These tests should, as mentioned in section 6.4.1, include different implementations of a join. In fact, one of these implementations might even use a commercial data base simply for the purpose of joining the different data sources.

An open issue is still the creation of meta data- and mapping files. While the “Meta Data Creator” presented in section 6.3 shows a possible path towards a semi-automated and machine-supported creation of the meta data used for the wrapper (wrapper.xml, section 4.2), it is not clear how the mapping (mediator.xml, section 4.1) and the ontology can be generated automatically.

A more formalized approach is needed to compare the MDDQL approach with the LAV- and GAV approaches. As already mentioned in section 3.1, more research is needed on the expressiveness and flexibility as well as the complexity and exactness of MDDQL.

The integration of this approach with other approaches in data integration needs to be considered. Furthermore, the creation of the Semantic Web needs to be followed closely and the inclusion into it must be a goal for MDDQL. In addition, a more distributed peer-to-peer (P2P)-version might be considered as does a net-
work of MDDQL-systems.

A new class of problems arises when an input device for distributed heterogeneous databases is addressed. This thesis benefited a lot from the fact that commitment- and synchronization problems between databases are not affected in a read-only environment.
8 Acknowledgments

I warmly thank Dr. Epaminondas Kapetanios for guiding me through this thesis. I am deeply grateful to Prof. Dr. Gustavo Alonso to accept me as a diploma student and always giving me time when needed. It was within a few hours, that his assistant Angela Nicoara had the test databases running on a different machine when things turned to the worst.

Also, I would like to thank Mia to bear me all day long at home and support me whenever needed. Sam and Ilari receive my thanks for helping me with English corrections. For theoretical questions, Björn was my primary addressee.

Last but not least, I’m very thankful to my parents for having supported me financially and morally and having made my studies possible.
References


[33] A.M. Ouskel and A. Sheth. Semantic interoperability in global information systems: A brief introduction to the research area and the special section.


List of Figures

1. The “big picture” of the system architecture. ................................. 9
2. A graphical RDF-representation of the wrapper description file. .... 15
3. A query in the MDDQL-querying tool ........................................... 26
4. A high-level query tree ................................................................. 27
5. The high-level query tree for several data sources without inter-
   section after splitting ................................................................. 27
6. The high-level query tree for different overlapping data sources ... 28
7. Several MIDs are mapped on more than one SMS within the same
   data source ................................................................................. 29
8. A M-Join of 3 tables step-by-step ................................................. 31
9. References between data sources .................................................. 34
10. “Shortest path” to connect data sources ....................................... 34
11. Meta data creator: Add attribute .................................................. 38
12. Meta data creator: Create terminology ....................................... 39
13. Meta data creator: Create references ......................................... 40
14. Packages overview ...................................................................... 41
15. UML-Diagram: mediator .............................................................. 42
16. UML-Diagram: wrapper ............................................................... 43
17. Time in milliseconds for M-Joins .................................................. 44
List of Tables

1  Aspects of the PSP/IQ model in [19] . . . . . . . . . . . . . . . . 20
2  Mapping the IQ dimensions into the PSP/IQ model as in [19] . . 21
3  Test system configuration. . . . . . . . . . . . . . . . . . . . . . 45
List of Algorithms

<table>
<thead>
<tr>
<th></th>
<th>M-⊗ Part 1: Fusion</th>
<th>30</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>M-⊗ Part 2: Shift</td>
<td>32</td>
</tr>
</tbody>
</table>