Universität Passau
Fakultät für Informatik und Mathematik

type of work

Implementierung einer verteilten Umgebung von medizinischen Datenquellen

David Goeth

13. März 2016

type of work
am Lehrstuhl für verteilte Informationssysteme
der Fakultät für Informatik und Mathematik
der Universität Passau

Erstgutachter: Prof. Dr. Harald Kosch
Betreuer: Armelle Natacha Ndjafa

Abstract

This is an abstract.

Contents

List of Tables vii

List of Figures ix

1 Introduction 1
1.1 Motivation . . . 1

2 Theoretical 3
2.1 Information Integration . . . 3
2.1.1 Architectures . . . 4
2.2 Dataspaces . . . 6
2.2.1 Query answering . . . 10
2.2.2 Towards Realization of Dataspaces . . . 13
2.2.3 Uncertainty in Data integration . . . 17
2.3 Towards a Model for Multimedia Dataspaces . . . 19

3 Reference Projects 21
3.1 DebugIT . . . 21

4 Implementation 23
4.1 Setting up databases . . . 23
4.2 Wrappers . . . 23
4.3 Register . . . 23
4.4 Basic dataspace querying . . . 23
4.5 Advanced query techniques/features . . . 23
4.6 GUI . . . 23
4.7 Validation . . . 23

5 Glossary 25

Bibliography 28

List of Tables

2.1 Kinds of heterogeneity 4

List of Figures

2.1 Environment of a Dataspace 14
2.2 System architecture 16
2.3 Architecture of a data integration system that handles uncertainty . . 19

3.1 The layered mediator architecture of the DebugIT project 21

1 Introduction

1.1 Motivation

2 Theoretical

2.1 Information Integration

Information integration addresses to integrate existing databases. Synonyms for information integration are information fusion, data consolidation, data cleansing or data warehousing. The goal of information integration is almost always to simplify the access to a range of existing information systems through a central, integrated component with a unified interface for users and applications. Therefore, integrated information systems provide a unified view on the data sources. Existing information systems can be diverse: Classical relational database systems, files, data accessed by web services or HTML formulas, data generating applications or even other integrated information systems. (from [1, p. 3-4], own translation).
Today, data is classified into three diverse categories. Structured data like it is stored in relational databases, have a predefined structure through a schema. Semi-structured data also have a schema, but they can deviate from the schema. An example of semi-structured data is a XML file without an accompanying XML schema. The third class is unstructured data and contains, as the name implies, no given structure. Typical unstructured data is natural language text. (from [1, p. 17], own translation).
If we speak of diverse information systems we usually mean heterogeneous systems. Heterogeneity exists among data sources as well as between data sources and the integrated system. In most of all integrated information systems only the latter heterogeneity matters, as data sources often do not communicate among themselves. To bridge heterogeneity, it is obviously necessary to translate queries and to implement missing functionality in the integrated system. Table 2.1 shows an overview of existing kinds of heterogeneity (from [1, p. 60/61], own translation).

In general, there are two different types of information integration: The materialized integration and the virtual integration. The difference between these two approaches is as follows: At materialized integration the data to be integrated is stored into the integrated system itself, so on a central point. The data in the data sources remains but for querying the materialized view is used. At virtual integration, the data is only transported from the data source to the integrated system while the query processing. This temporary data is then again discarded. So integration isn't done once but on each query. Of course, an integrated information system can use both principles. Such a system is called hybrid. Both types have in common that a query is processed on a global schema. For the virtual integrated system, the data only exists virtual, thus relations between data sources and the global schema have to be specified and on query time the query has to be split into into query schedules. The schedules are responsible to extract the needed information from the different data sources and subsequently merge and transform the data (from [1, p. 86-88], own translation).

Table 2.1: Kinds of heterogeneity

Technical heterogeneity	includes all problems to realize the access of the data of the data sources technically. This heterogeneity is overcome if the integrated system is able to send a query to a data source and that data source principally understands the query and produces a set of data as result.
Syntactic heterogeneity	includes problems in the illustration of information. This heterogeneity is overcome if all information meaning the same are illustrated equally.
Data model heterogeneity	means problems in the presentation of data of the used data models. This heterogeneity is solved if the data sources and the integrated system use the same data model.
Structural heterogeneity	includes differences in the structural representation of information. This heterogeneity is solved if semantic identical concepts are also structural equally modeled.
Schematic heterogeneity	Important special case of the structural heterogeneity, whereby there are differences in the used data model.
Semantic heterogeneity	Includes problems regarding the meaning of used terms and concepts. This heterogeneity is solved if the integrated system and the data source understand by the used names for schema elements really mean the same. Equal names means consequently equal meaning.

2.1.1 Architectures

Integrated systems evolved from distributed databases which in turn is based on the classical monolithic database. In the following basic database concepts, on which integrated systems base, and the different types of integrated systems are presented shortly.
The **monolithic database** is subdivided into three layers: The *internal (or physical) view* is responsible for the storage of the data on the respective database. One layer upwards comes the *conceptual (or logical) view* modeling the data on a conceptual way. The conceptual schema defines which data model is used, which data is stored in the DBMS and the relations among the data. Splitting the data into the internal and conceptual view, makes the data independent from the physical storage medium. The top layer is called *external (or export) view* and is being concerned with modeling of data as well as the conceptual view. However it isn't modeled the whole application domain. The external view only specifies which part of the conceptual schema is provided to the respective application. An export schema is initially a subset of the conceptual schema, but can be transformed and aggregated. In the external view are defined access restrictions, too. Splitting up conceptual and external view ensures *logical data independence* ([1, p. 84/85], own translation).

image of monolithic database structure

Next comes the **distributed database** architectures. The idea is to distribute the data onto several systems (physical and logical), but a user should be able to query

all data at once. In order to achieve this, the architecture is subdivided into *four layers*. Each data source owns a local internal and local conceptual schema. The latter only mirrors the data managed by the local database. On top of the local conceptual schemes stands a global conceptual schema. This schema models the whole application domain and is the central point of reference for the external schemes playing the same role as in the three-layers-architecture. Distributed databases are close coupled and thus heterogeneity problems don't occur ([1, p. 91-93], own translation).

image of distributed database structure

For allowing to connect heterogeneous databases, **Multidatabase Systems** (MBS) have been evolved. MBS are collections of autonomous databases being loosely linked. Each database grants external applications access to its data. The access is done using a database language which allows to query several databases in one query. Such a language is called multidatabase languge. To obtain the autonomy of the involved databases, a MBS has no global conceptual schema. Instead, each local database keeps an export schema defining which part of the local conceptual schema is provided to external applications. It is assumed that no data model heterogeneity is contained in a MBS, i.d. all databases use the same data model or either the multidatabase language or the local data source provide a translation to the global data model. Now, each application can create its own external schema, which integrates one or more data sources. So its the task of the application doing the integration task. A MBS provides only a suitable language for querying ([1, p. 93/94], own translation).

image of MBS database structure

In contrast to MBS, **federated database management systems** (FDBMS) have a global conceptual schema. This schema is the central and stable point of reference for all external schemes and there applications. But in contrast to distributed databases the global schema results after the local schemes with the goal to provide a integrated view of existing and heterogeneous data sets. Data sources keep a high degree of autonomy. The used data model in the global scheme is known as canonical data model. Every local export schema has to be mapped to this global schema. Thus, in order to support heterogeneity in the data model, another layer is needed between local conceptual schema schema and the local export schema. This new layer is called *local component schema* and translates the local conceptual schema into the canonical data model. As a result, FDBMS consists of five different layers. The global schema can be created either by schema integration or schema mapping. Both variants are introduced later (TODO) ([1, p. 94/95], own translation).

image of FDBMS database structure

Mediator-based information systems are a generalization of the previous architectures, as they know only two separate components, namely Wrappers and

Mediators: Wrapper are software components responsible for the access to a solely data source. A Wrapper has to break down technical, data model, schematic and interface heterogeneity. Mediators access one or more wrappers and deliver a specific value, normally structural and semantic data integration. In this architecture, data sources usually don't know of the existence of the integrated system, and so autonomy is preserved for all data sources ([1, p. 97], own translation).

image of Mediator-based information system structure

In **peer data management systems** (PDMS) there is no separation between data source and integration system. Query can be posed from every system to every other system within the integrated system. The other system than tries to calculate responses with own or other sourced data. So each participant of the integrated system, a so-called peer, is a mediator and a data source at the same time.

Ontology-based integration is an approach of semantic integration using ontologies. This approach assumes that the discourse range is able to be specified so exact, that semantic heterogeneity can be solved in a formal model by logical inference. Hereby, the formal model is called ontology. Ontologies define the vocabulary describing all concepts of the field of application, and the relations between these concepts. At the same time, an ontology often serves as global schema of the integration layer. For specifying ontologies, special classes of logics are used, the so-called description logics. With description logics, classes of a domain and relations among them can be specified much more precise than with the relational data model. ([1, p. 267], own translation).

2.2 Dataspaces

Dataspaces describes a new abstraction of data management [3]. In current scenarios it is rarely the case, that data to be managed is solely stored in a convenient relational Database Management System (DBMS) or in another, single data model. Therefore, developers often face the challenge to deal with heterogeneous data on a low level base. Challenges are to provide search and query capabilities, enforcing rules, integrity constraints, naming conventions etc.; tracking lineage; providing availability, recovery and access control; and managing evolution of data and metadata. That issues are ubiquitous and arise in enterprises, government agencies and even on one's PC desktop.

As a response to this problem, the authors postulate the concept of dataspaces and corresponding to this, the development of DataSpace Support Platforms (DSSPs). Shortly said, the latter provides an environment of cooperating Services and guaranties, that enables software developers to concentrate on their specific application problem rather than taking care of returning issues in consistency and efficiency of huge, linked but heterogeneous data sets. The remarkable properties of a dataspace system are defined as follows:

A DSSP must deal with data and applications in a variety of file formats, that are accessible through many systems with different interfaces. A DSSP has to support all kinds of data in the dataspace rather than only a few (as DBMSs do). Although a DSSP provides a integrated possibility for searching, querying, updating and administration, the data often can only be accessible and modifiable through native interfaces. Therefore DSSPs haven't full control over their data. Queries on a DSSP may offer varying levels of services. In some cases the answers can be approximated resp. best-effort. An example: If some data sources are unavailable for some reasons the DSSP is able to return the best result as possible. Therefore it uses the data that are available at the time of the query. A DSSP has to provide the tools that allow a tighter data integration process in the dataspace as necessary.

Many of the services a dataspace provide, data integration and exchange systems provide, too. The main difference between these systems is that data integration systems need a semantic integration process before they can provide any services on the data. But dataspace is not kind of a classic data integration approach. In order to avoid semantic integration, a dataspace uses the concept of data coexistence. The idea is to provide base functionality over all data sources, regardless of their specific integration constraints. E.g. a DSSP is able to provide a keyword search similar to a desktop file search. If more sophisticated operations are required such as relational queries, data mining or monitoring of specific data sources, additional effort can be done to integrate the sources tighter. This incremental process is also called as "pay-as-you-go" fashion.

In chapter 3 of the paper the authors pottered at the logical components and services a DSSP should have:

Logical components

A Dataspace should contain all information being relevant for a specific organization/task regardless of their file format or storage location, and it should model a collection of relationships between the data repositories. Therefore the authors define the dataspace as a set of participants and relationships. Participants of a dataspace are individual data sources . Some participants support expressive query languages for querying while others only provide limited access interfaces. Participants can reach from structured, semi-structured right up to unstructured data sources. Some sources provide traditional updates, some are only be appendable, while others are immutable. Further, dataspaces can be nested within each other, which means that a dataspace should be able to be a part of another dataspace. Thus, a dataspace has to provide methods and rules for accessing its sub dataspaces.

Services of dataspaces

The most basic services a dataspace should contain is the cataloging of data elements of all participants. A catalog is an inventory of data resources containing all important information about every element (source, name, storage location inside the source, size, creation date, owner, etc.)of the dataspace. It is the infrastructure for the most other dataspace services. Search and query are two main services a DSSP must provide. A user should be able to state a search query and iteratively precise it, when appropriate, to a database-style query. For the dataspace approach it is a key tenet that search should be applicable to all of the contents, regardless of their formats. The search should include both data and meta-data. The user should be

enabled to discover relevant data sources and inquire about the completeness, correctness and freshness. A DSSP in fact should be aware of the gaps in its coverage of the domain. A DSSP should also support updating data. Of course, the mutability of the relevant data sources determines the effects of updates. Other key services would be monitoring, event detection and the support for complex workflows (e.g. it is desired that a calculation is done if new data arrives and that the result will be distributed over a set of data sources). On a similar way a DSSP should support various forms of data mining and analysis. Not every participant will provide all necessary interfaces for being able to support all DSSP features. Hence it is necessary, that data sources can be extended on various ways. E.g. a source don't store its own meta-data, so there has to be an external meta-data repository for it.

Components and architecture of a dataspace system

Catalog and Browse Information about all participants and the relationships among them are stored in the catalog. It must also deal with a large variety of sources and supports to provide different levels of information about their structure and capabilities. It is important, that the catalog includes the schema of the source, statistics, rates of change, accuracy, completeness query answering capabilities, ownership, and access and privacy policies for each participant. Relationships may be stored as query transformations, dependency graphs or even textual descriptions. If possible, the catalog should include a basic inventory of the data elements at each participant: identifier, type, creation data and so forth. Then, it can support basic browsing over the participant's inventories. It isn't a very scalable interface, but it can be used to response to questions about the presence or absence of a data element, or determine which participants have documents of a specific type. On top of the catalog, the DSSP should have a model-management environment allowing the creation of new relationships and manipulation of existing ones.

Search and Query The component has to offer the following capabilities:

(1) *Query everything:* Any data item should be queryable by the user regardless of the file format or data model. Keyword queries should be supported, initially. When more information about a participant is collected, it should be possible to gradually support more sophisticated queries. The transition between keyword query, browsing and structured querying should be gracefully. And when answers are given to keyword (or structured) queries, the user should be able to refine the query through additional query interfaces.

(2)*Structured query:* Queries similar to database ones should be supported on common interfaces (i.e. mediated schemas) that provide access to multiple sources or can be posed on a specific data source (using its own schema). The intention is, that answers will be obtained from other sources (as in peer-data management systems), too. Queries can be posed in a variety of languages (and underlying data models) and should be translated into other data models and schemas as best as possible with the use of exact and approximate semantic mappings.

(3) *Meta-data queries:* It is essential, that the system supports a huge variety of meta-data queries. These include (a) source inclusion of an answer or how it was derived or computed, (b) timesteps provision of the data items that are included in the computation of an answer, (c) specification of whether other data items may depend on on a particular data item and the ability to support hypothetical queries.

A hypothetical question would be 'What would change if I removed data item X?. (d) Querying the sources and degree of uncertainty about the answers. Queries locating data, where the answers are data sources rather than specific data items, should be supported, too.

(4) *Monitoring:* All stated Search and Query services should also be supported in an incremental form which is also applicable in real-time to streaming or modified data sources. It can be done either as a stateless process, in which data items are considered individually, or as a statefull process. In the latter multiple data items are considered.

Local store and index A DSSP has a store and index component to achieve the following goals: to create efficiently queryable associations between data items in different participants. Important is here, that the inde should identify information across participants when certain tokens appear in multiple ones (in a sense, a generalization of a join index) to improve accesses to data items with limited access patterns. Here, the index has to be robust in the face of multiple references to real-world objects, e.g. different ways to refer to a company or person. to answer certain queries without accessing actual data sources. Thus the query load is reduced on participants which cannot allow ad-hoc external queries. to support high availability and recovery

The index has to be highly adaptive to heterogeneous environments. It should take as input any token appearing in the dataspace and return the location at which the token appears and the roles at each occurrence. Occurrences could be a string in a text file, element in file path, a value in a database, element in a schema or tag in a XML file.

Discovery Component This components locates participants in a dataspace, creates relationships between them, and helps administrators to refine and tighten these relationships. For each participant the component should perform an initial classification according to the participant's type and content. The system should provide an environment for semi-automatically creating relationships between existing participants and refining and maintaining existing ones. This involves both finding which pairs of participants are likely to be related, and then proposing relationships which a human can verify and refine. The discovery component should also monitor the content in order to propose additional relationships in the dataspace over time.

Source Extension Component Some participants may don't provide significant data management functions. For example, a participant might be no more than departmental document repository, perhaps with no services than weekly backups. A DSSP should support to enrich such a participant with additional capabilities, such as a schema, a catalog, keyword search and update monitoring. It may be necessary to provide these extensions locally as there can be existing applications or workflows that assume the current formats or directory structures. This component also supports "value-added" - information held by the DSSP, but not present in the initial participants. Such information can include "lexical crosswalks" between vocabularies, translation tables for coded values, classifications and ratings of documents, and annotations or linked attached data set or document contents. Such information must be able to span participants in order to link related data items.

Following paragraph have to be moved to more suitable place!
The document "Principles of Dataspace System" of Alon Haley, Michael Franklin and David Meier[4] is based on their previous work "From Databases to Dataspaces - A New Abstraction for Information Management"[3] where the concept of dataspaces was firstly presented, and poses specific technical challenges realizing Dataspace Support Platforms (DSSPs) relating to query answering, introspection and the benefits of human attention for improving semantic relationships within a dataspace. In the following are stated the results of the above mentioned challenges:

2.2.1 Query answering

Queries are posed in a wide range of different languages. Most of the activities will properly begin with a keyword search, but it will also be common to see queries as a result of a form (which results to queries with several selectable predicates). It will come to more complex queries when a user interacts deeper with a certain data source. If it isn't explicitly stated, it is usual, that a user is likely to believe that a query considers all relevant data in a dataspace, regardless of the used data model or schema. Even if a query is posed to a data source, it is implicitly expected that the system considers the data of other sources, as well. If additional answers are desired, one has to do transformations on the schema and the data model.

Challenges of answer querying

Answers corresponded to queries of a dataspace are different from traditional queries in several ways. The challenges to it are analyzed more deeply in the third chapter:

Ranking: Queries are typically sorted by their relevance, similar to a web search engine. Ranking is necessary not only for keyword search, but also for structured queries, when transitions to other data sources should be approximated.

Heterogenity: Answers will come from many sources and will differ in their used data model and schema. The Ranking has to manage heterogeneity, too.

Sources as answers: In addition to base elements (e.g. documents or tuples), a DSSP should be able to provide sources, as well. This means that it returns links to locations where additional answers can be found.

Iterative queries: Normally, the interaction with a dataspace can't be reduced to the process of posing a sole query and getting an answer to it. Instead, a user is involved in an information finding task that requires a sequence of queries, each being a refinement or modification on the previous ones.

Reflection: It is expected, that a DSSP reflects on the completeness of its coverage and the accuracy of its answers.

Dealing with challenges

The first step to face these challenges is to build a formal model. Therefore, the authors propose five directives:

Challenge 3.1. The development of a formal model for analyzing the query answering in a dataspace.

Sub-Challenge 3.2. The development of intuitive semantics to answer a query taking into consideration a sequence of earlier posed queries.

Sub-Challenge 3.3. The development of a model for an information finding task which includes operations on lower levels.

Sub-Challenge 3.4. The development of algorithms sorting the data sources according to how likely they are to contain the answer. These algorithms should start from a keyword query and operate on a large collection of data sources.

Sub-Challenge 3.5. The development of methods for ranking answers retrieved by multiply heterogeneous data sources.

Query answering model and challenges

Challenge 3.7. The development of techniques to answer queries based on the following ideas or combinations of it:

- apply several approximate or uncertain mappings and compare the answers obtained by each.

- apply keyword search techniques to obtain some data or some constants that can be used in instantiating mappings.

- examine previous queries and answers obtained from data sources in the dataspace and try to infer mappings between the data sources. Whenever we have access to queries that span multiple data sources, try to infer from them how the sources are related (e.g., the join attributes should provide some hint of common domains).

- Develop a formal model for approximating semantic mappings and for measuring the accuracy of answers obtained with them.

- Develop automatic best-effort methods for translating a query over one data set onto the other.

Introspection

In chapter 4 introspection within a dataspace is analyzed. Introspection describes on this occasion the ability of a dataspace to observe assumptions and uncertainties and specifying their origin. In contrast to traditional databases introspection isn't a nice feature but a necessity. Introspection has to be possible over the following highly related parameters: Lineage, Uncertainty and Inconsistency. Thus, one speaks of LUI introspection in the context of dataspaces.

Uncertain databases

Uncertainty arises in applications for data management if the exact state of the (data) world is not known. The goal of a uncertain database is to represent a set of possible states the world can have. Usually, the states are referred to as possible worlds. Every possible world represents a complete valid state of the database. For declaring these states there were developed several formalisms [5, 6, 7, 8, 9]. Examples for formalisms would be a-tuples, x-tuples or c-tables[10].

Inconsistency in databases

The task of an inconsistency database is to handle situations where the database contains conflicting data. A common example would be the existence of two values of

a salary of an employee whereby the values are obtained from different data sources. The essence for solving this problem is to consider all possible repairs. A repair is a minimal change resulting the database to a consistent state. As a rule, there are more one one possibility which is why the database owns several repairs to choose from. That's also the reason for the tight relationship between uncertainty and inconsistency as inconsistency can be interpreted as uncertainty over the knowledge, which of the conflicting value is the best solution.

Modeling of data lineage

The lineage of a tuple reports how the tuple was derived from a certain data set. There is a distinction to internal and external lineage. Internal lineage applies to tuples of a query result – lineage specifies here how the tuple was derived within a database. External lineage relates to tuples inserted into a database, the lineage relates hereby to the external sources or processes by which the tuples were inserted.

LUI Introspection

A DSSP should provide a uniform mechanism for modeling uncertainty, inconsistency and lineage. Thereby the following challenges arise:

Challenge 4.1. Develop a formalism for enabling the modeling of uncertainty, inconsistency and lineage.

Sub-Challenge 4.2. Develop a formalism which captures the uncertainty over general forms of inconsistency in databases.

As inconsistency results to a special type of uncertainty, the formalism for uncertainty should leverage this special structure. The formalism for uncertainty tells us only what the possible states of the world are and sometimes it assigns every possible world its corresponding possibility. But in many cases, it is the only way to resolve uncertainty and to get knowledge about data lineage and how and where the data was derived from.

Web search engines already unify uncertainty and lineage in a simple way. One of the main reasons why the authors want to unify lineage and uncertainty is to reason the relationship between external sources and their impact on answers. Thereby arises the following challenges:

Sub-Challenge 4.3. Develop a formalism which represents the external lineage and reasons it.

To combine uncertainty and lineage, there exists

Sub-Challenge 4.4. Develop a general technique for extending every kind of uncertainty and for investigating the representative and calculative advantages of this procedure.

To achieve this, the object that should be assigned uncertainty, has to match to the object lineage is attributed to.

Uncertainty on Views

There's a general problem with modeling uncertainty as the uncertainty formalism associate uncertainty with a single schematic construct: tuples in the case of x-tuples and attribute values in the case of a-tuples. So the chose of database schema and

normalization limits the kinds of uncertainty to express. In the case of using views, the authors propose to link uncertainty with the view's tuples, hence:

Sub-Problem 4.5. Develop formalisms where uncertainty can be attached to tuples in views and view uncertainty can be used to derive uncertainty of other tuples

Finding the right answer

This chapter address to what has to be done to determine the quality of a query answer (how good is a query?).

Sub-Challenge 4.6. Define metrics for comparing the quality of query answers and answer sets over a dataspace, and find efficient techniques to process queries.

A limited version of this issue was already addressed with minimal repairs related to inconsistent databases. Here it will be tried to construct a consistent database as tight as possible on the base of the inconsistent one. For improving the results, it is necessary to do some preferences, which results in the following challenges:

Sub-Challenge 4.7. Develop query-language extensions and their corresponding semantics that enable specifying preferences on answer sets along the dimensions of completeness and precision, certainty and inconsistency, lineage preferences and latency.

Along with the specification of preferences there are needed methods for reasoning query answer sets. This is essential for comparing answer sets. Query containment [11] will be extended in the context to dataspaces. By which the following challenge submits to:

Sub-Challenge 4.8. Define notions of query containment that take into consideration completeness and precision, uncertainty and inconsistency and lineage of answers, and efficient algorithms for computing containment.

The following issues stands in relation to section 4.1. of the paper:

Sub-Challenge 4.9. Develop methods for efficient processing of queries over uncertain and inconsistent data that conserve the external and internal lineage of the answers. Study whether existing query processors can be leveraged for this goal.

2.2.2 Towards Realization of Dataspaces

In the paper "Towards Realization of Dataspaces", written by Ibrahim Elsayed and Peter Brezany[12], the architecture and functionality of a Dataspace Management System(DMS) is presented. Additionally it is discussed, how grid technology of future implementations is able to support such an architecture. A DMS is a set of software programs, which controls the organization, storage and retrieval of data in a Dataspace. It also handles the security and integrity of a Dataspace. In the following, the architecture and the functionality requirements of a DMS will be described.

Requirements

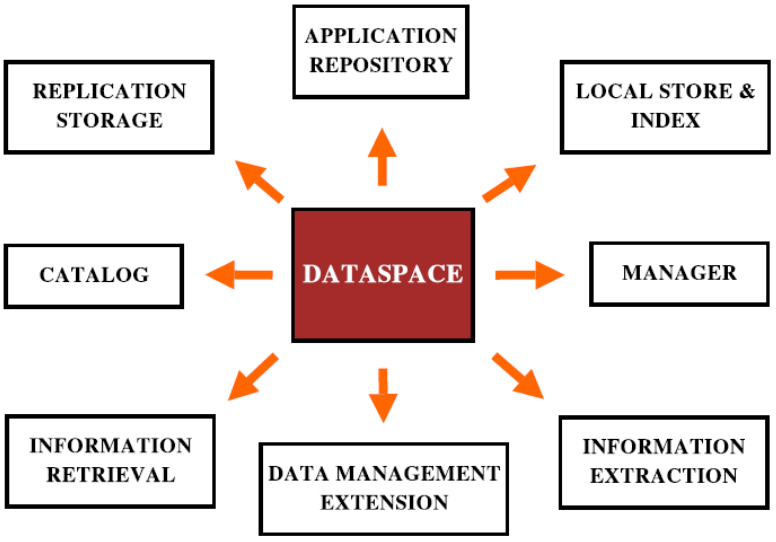

Figure 2.1: Environment of a Dataspace

Information Retrieval: Queries and searches are two different methods of information retrieval and represent together one of the main services a DMS should support. Generally, queries and searches should be supported by all participants of the Dataspace, regardless of their used data model. It shouldn't make a difference for a user to operate on a sole database or on a Dataspace. A good known and simple search operation is keyword searching. The Support of spanning such a search method over all participants, is a challenging research topic. The development of methods for keyword searching on relational and XML databases was done by the data-engineering community [13, 14, 15], For supporting a global query functionality allowing to formulate uniform queries on all Dataspace participants, intelligent methods for interpreting and translating of queries in several languages are required. Methods for query translation were investigated by a large body of research communities [16, 17].

Information Extraction: The world-wide web has become a large data container, by now. For allowing to postprocess data obtained from the web, techniques for web information extraction, extracting relevant information from semi-structured web pages should be supported. For further processing, extracted content has to be converted to structured data and saved locally. Non structured web documents have to be classified first using text mining techniques, which organize the parsed documents into groups with the help of ontologies [18]. Based on these ontologies, a keyword search is possible to find individual documents. Structured documents allow easier access and integration due to the rich semantically information included in the data representation.

Data Management Extensions: This component provides possibilities to improve low-level working Dataspace components. All these base components have only limited data management capabilities. It is a task of a DMS to provide additional functionality such as backup, recovery and replication.

Catalog: Contains a detailed description of all participants of the Dataspace. Beside basic information about the participants, such as owner, creation date, etc., the description for each participant should also contain semantic information about its

data. A user should be able to browse the catalog for getting more detailed information about certain data sources. The catalog may reference a meta-data repository to separate the basic and more detailed descriptions.
Manager: In order to realize the above mentioned functionality, a central component managing the system and interacting with the user is needed. Alongside user authentication, right assignments and other services, this manager component is responsible for communicating with all participants. Thus, this component serves as an interface between the users and the participants of the Dataspace.
Local Store and Index: This component is responsible for caching search and query results, so that certain queries can be answered without the need of accessing the actual data sources. Furthermore it supports the creation of queryable associations between the participants.
Replication Storage: Allows to copy participant data in order to increase its access time. This results in high availability and high recovery is supported.
Application Repository: With this component, the user is able to share data analysis tools, domain specific models, evaluations, etc., which can be applied to the (available) data of the Dataspace.

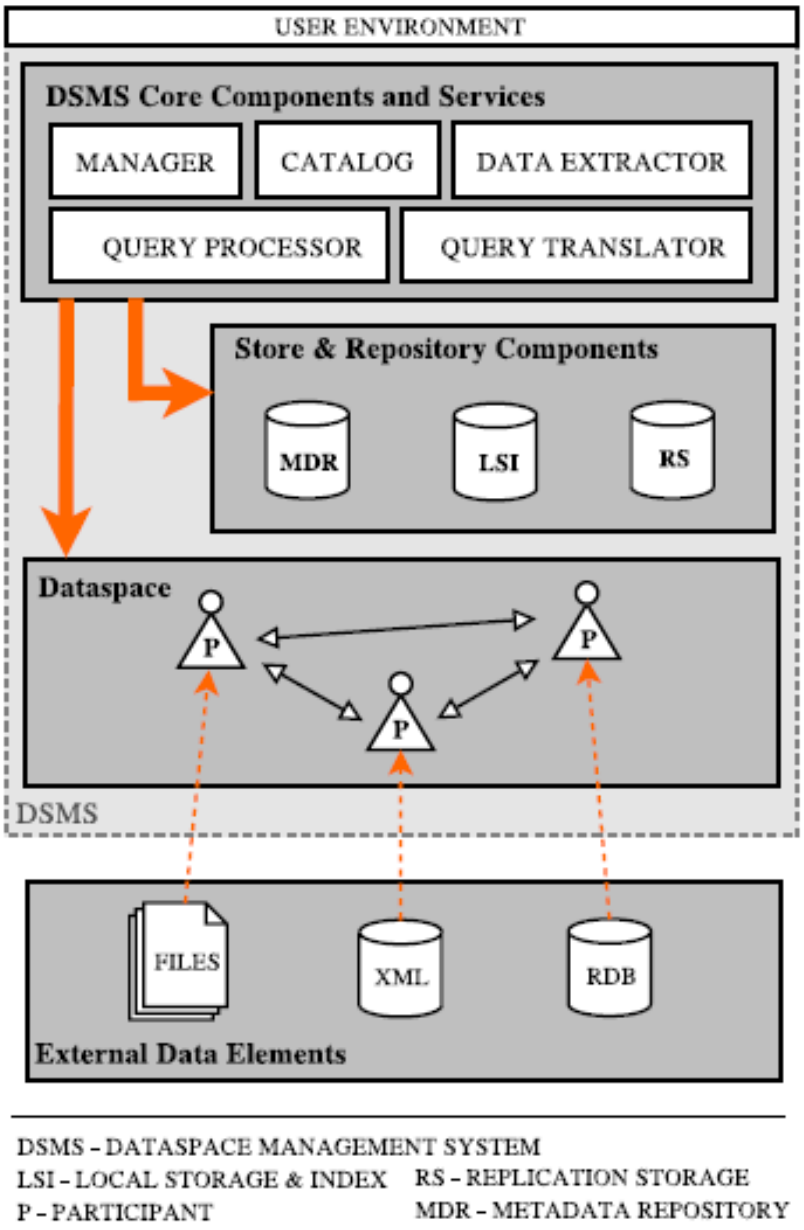

Figure 2.2: System architecture

Generallly, a Dataspace consists of Dataspace components, the so-called participants, and relationships between the participants. Components are individual data sources, such as relational databases, text databases, XML databases, data stream systems, web services or other data archives.
A Dataspace component includes a description for describing what data it contains, which storage formats are used and which query mechanisms are supported. In addition it contains information about the storage location of the data and whether the data was replicated, equally it contains information about relationships to other participants. The information and data are marked if they are added by the management system, the user or were automatically generated.
Each participant is described the same way as the components and is subsequently registered within the catalog. Thus they are available as data sources for a more

abstracted layer, which provides global query and search methods. This global layer is embedded in the DMS. It supports the user to create new Dataspace components, to add relationships between other participants, to register participants within the catalog, to browse the catalog for available participants, to decide whether to share all or only selected participants within a community including assignments of rights, and to search all or only selected participants of a Dataspace.
As each participants is managed by its own and thus is responsible for its data management such as data update, recovery and replica, it isn't a task of the DMS to change the data represented by the participants. Therefore, the DMS has no administration privileges for writing and changing content of the data of a participant. But the DMS provides possibilities to add missing data management services, such as data replication, transformation, download, upload, etc..

Managing Sub-Dataspaces
The idea behind a Sub-Dataspace is to create a new Dataspace that is included by a another and thus is a subset of a bigger Dataspace. This functionality is needed for access privileges and security management. An administrator of a Dataspace is thus able to define different data access rights for certain groups. Following paragraph have to be moved to more suitable place!
In the paper "Data Modeling in Dataspace Support Systems" [25], the authors face the issue of uncertainty in dataspaces. In their view, a dataspace needs to model uncertainty in its core. In their work, they described the concepts of probabilistic mediated schemas and probabilistic mappings as enabling concepts for DSSPs. With this foundation, it is possible to completely bootstrap a pay-as-you-go integration system. Following paragraph have to be moved to more suitable place!
Current data integration systems are essentially a natural extension of traditional databases in that queries are specified in a structured form and data is modeled in one of the traditional data models like relational or XML. The System for data integration also has exact information about how the data in the sources map to the schema used for integration. So, for data integration there is a need for large upfront effort in creating the mediated schema and the schema mappings. Dataspace Support Platforms (DSSP) shall considerable reduce this upfront effort. The system should be able to bootstrap itself and provide useful services with no human intervention. When the data management needs become clearer (through user feedback or as sources are added), the system evolves in a pay-as-you-go fashion. In order to develop DSSPs, it is impossible to rely on the same data modeling paradigms data integration systems use. One cannot assume, that the mediated schema is given in advance and that the schema mappings between the sources and the mediated schema will be accurate. Therefore the authors argue that uncertainty has to be considered in the mediated schema and in the schema mappings.

2.2.3 Uncertainty in Data integration

Sources of Uncertainty

Uncertain mediated schema: The set of schema terms in which queries are posed is called the mediated schema. Not necessarily they cover all the attributes in any source, it cover rather the aspects of the domain that the developer of the application wishes to expose to the user. For several reasons uncertainty happens

in the mediated schema. First, if the mediated schema is derived from the data sources during bootstrapping, there will be some uncertainty about the results. When domains are broad, there will be also some uncertainty about how to model them, because in general, there will be overlappings in different topics.

Uncertain schema mapping: Schema mapping defines the semantic relationships between the terms in the sources and the terms used in the mediated schema. Although, schema mappings can be inaccurate. In a dataspace, many of the initial schema mappings are properly automatically derived, and therefore they may inaccurate. In general, it is impossible to create and maintain precise mappings between data sources.

Uncertain data: Some of the data may be obtained by automatism as data sources may not always be structured well. Additionally, unreliable or inconsistent data may be contained in systems with many sources.

Uncertain queries: Properly, much of the early interaction with a dataspace will be done through keyword queries as the users aren't aware of a (non-existent) schema. The queries have to be translated into some structured form so they can be reformulated with respect to the data sources. At this point, multiple candidate structured queries could be generated and thus some uncertainty arises about which query captures the real intention of the user.

System architecture

The most fundamental characteristic of this system is that it is based on a probabilistic data model. In contrast to a traditional data integration system, which includes a single mediated schema and assumes a single (and correct) schema mapping between the mediated schema and each source, the data integration module of a DSSP attaches possibilities to multiple tuples, mediated schemas, schema mappings and possible interpretations of keyword queries posed to the system.

For DMS it is assumed that queries are posed as keywords. This is contrary to traditional integration systems which assume the query to be posed in a structured fashion(i.e. that it can be translated to some subset of SQL). So, a DSSP must first reformulate a keyword query into a set of candidate structured queries, before it can reformulate the query onto the schemas of the data sources. This step is also called keyword reformulation. It is important to note, that keyword reformulation differs from keyword search techniques on structured data (see [13], [15]) in that (a) it does not assume access to all data in the sources or that the sources support keyword search, and (b) it tries to distinguish different structural elements in the query in order to pose more precise queries to the sources. In any case, keyword reformulation should benefit from techniques that support answering search on structured data.

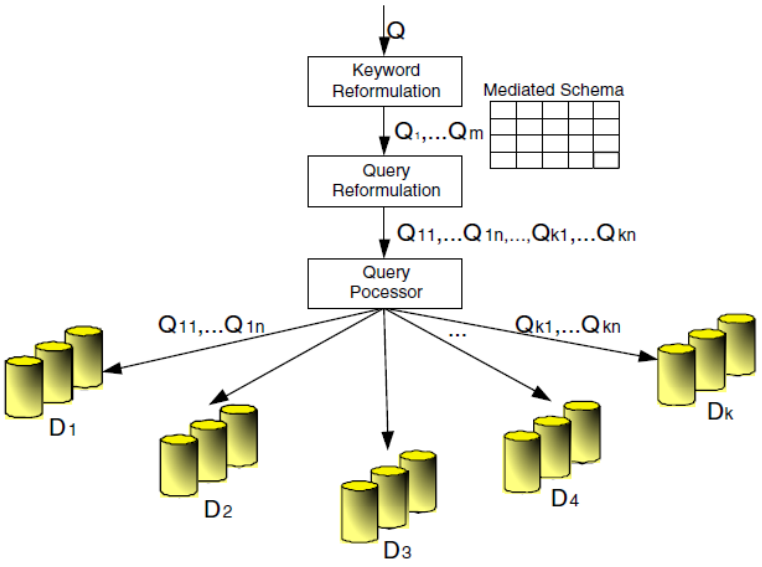

Figure 2.3: Architecture of a data integration system that handles uncertainty

Different is also the query answering in a DSSP. It doesn't find necessarily all answers on a given query, rather than typically find the top-k answers, and rank these answers most effectively. The architecture of the proposed system is shown in 2.3. The DSSP contains a number of data sources and a mediated schema (probabilistic mediated schemas are omitted here). When the user poses a query, which can be either a structured or a keyword query, the systems returns a set of answer tuples, each with a calculated probability. If a keyword query was posed, a keyword reformulation has firstly be done to translate it into a set of candidate structured queries on the mediated schema. Otherwise, the candidate query is the posed query itself.

2.3 Towards a Model for Multimedia Dataspaces

In the paper 'Towards a Model for Multimedia Dataspaces'[2] the authors present a representation model for dataspaces, which is an approach based on the dataspace and dataspace-view paradigm for uniformly represent structured and semi structured data, ontologies and other similar knowledge representation models. With that model, the authors address the current explosion of digital information and data sources. Many branches (i.a. in the medicine) need now large amounts of distributed, heterogeneous data.

However, traditional data integration methods are barely applicable to formulate search queries on a distributed, heterogeneous data set, containing files with many different file formats, as traditional data integration systems are designed for complete structured data. Thus, these systems need a global schema for calculating search queries. Addressing this issue, the concept of dataspaces was developed as an abstraction of wide-area, heterogeneous and distributed data management. For dataspaces there is no need of upfront efforts to semantically integrate data before basic services such as keyword search can be provided.

Dataspaces support uncertainties in schema-mapping and consider that schema mapping from sources to mediated schema may be incorrect. A further interesting feature of dataspaces is successive data integration. So, the system is able to integrate data in iterations as the time goes on depending on user needs.

Research in the field of dataspaces is currently very active, but despite of the deep interest, existing models suffer on a number of shortcomings limiting their applicability. These include the tendency of overlooking the different types of relations that can exist between data items, which restricts the amount of information they can generally integrate. Second, they focus on classical text data ignoring the specifics of multimedia data. Third they don't provide the fine-granularity in the persistence of integrated data that is needed in certain domains.

To address these issues, the authors developed a dataspace model, which sees the dataspace as a set of classes, objects (instances of classes) and relations. In the latter case there exist relations between classes (CRC), relations between objects and classes (ORC) and relations between objects (ORO). Furthermore relations can be internal or external defining whether the anticipating relation objects are in the same data source or in different ones.

A design goal of the model is the maximization of its expression in terms of the types of relations that it can represent, enabling it to deal with information originating from structured data, semi-structured data, ontologies and other forms of knowledge representation as well as from canonical knowledge.

Import to note is the fact that the model includes similarity relations in the type of relations. Similarity functions are a feature of multimedia data that defines a measurement for non-exact matching between objects. This kind of relations can be used to derive relations between other objects in the dataspace. Additionally the model introduces the concept of dataspace-view. This makes it possible to store query results of an existing dataspace into a new sub-dataspace with different modes of persistence(virtualized view, materialized view, mode of synchronization with the content of the original sources,...).

3 Reference Projects

3.1 DebugIT

The DebugIT project is a good reference for analyzing how medical data integration can be done though this project didn't use a dataspace approach but an ontology-mediated [19, 1] approach. DebugIT stands for "Detecting and Eleminating Bacteria Using Information Technology" and its main goal is to provide a platform for high-throughput analysis of distributed clinical data as a response to the spread of antibiotic resistance of infectious pathogens in European hospitals[20]. The team of the DebugIT project released the architecture of their system in [21]. In that paper it is stated, that the system uses an ontology-based approach for allowing antibiotics resistance data semantically and geographically interoperable. This makes it possible to integrate distributed clinical data EU-wide in real-time for monitoring antibiotic resistance. As well, the system is structured in four tiers and works in a service oriented manner (SOA). Figure 3.1 shows the layered architecture of the DebugIT's system:

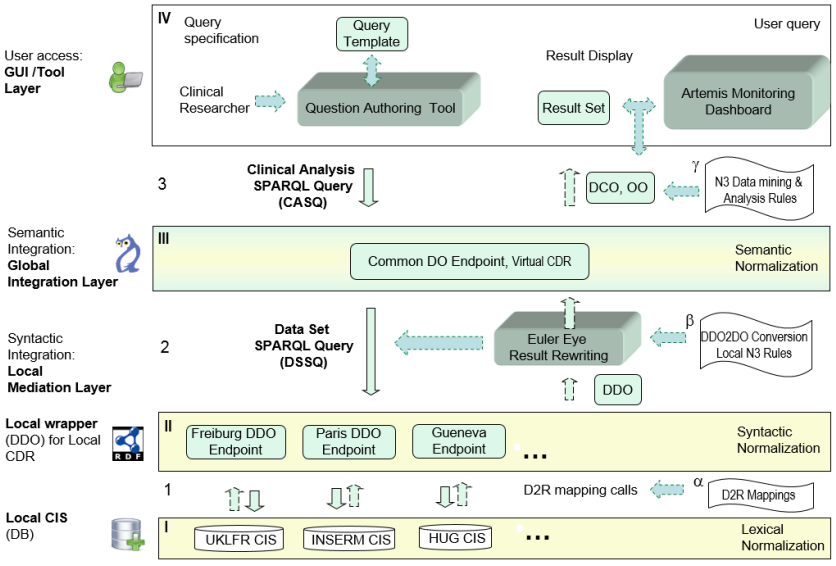

Figure 3.1: The layered mediator architecture of the DebugIT project

Altogether there are 3 data representation layers marked with the Roman Numerals I-III. The query flow between the data layers is specified with 1-3 and the corresponding mappings are given with the Greek letters α, $\beta$and γ.

On the first integration layer (I) relational data are lexical normalized by the use of mappings of medical terminology and morphosemantic mapping employed by the

Averbis Morphosaurus software[22]. Ambiguities can be enriched with ontological expressions (formulated in OWL) on the integration layers II and III. The layer II works with RDF data, wherefore relational data from the layer I is transformed to RDF through D2R mapping calls [1] on the first mediation layer (1). On the second integration layer information about the data and their corresponding vocabulary is stored in Data Definition Ontologies (DDOs) [23]. A DDO bridges a local data model and a semi-formal data model on the local mediation layer for integrating syntactic data and provides an Extract, Transform, Load (ETL)-process [2] for the next integration layer. For each Clinical Data Repository (CDR) such a DDO is locally created.

Layer II contains also a SPARQL endpoint, for allowing Layer III to query its data. These queries are specified through Data Set SPARQL Query (DSSQ). On the second mediation layer (2) the local DDO data is bind to the global DebugIT Core Ontology (DCO) [24], the data representation of layer III. The corresponding mapping (2) is done through DDO2DCO using the N3 language [3] and Simple Knowledge Organization Structure (SKOS) mappings [4]. The binding is done through the Euler Eye Reasoner [5].

Data layer III represents a virtual Clinical Data Repository (vCDR), which joins the local Clinical Information Systems (CISs). Through the virtualisation data of all CIS can be globally queried and privacy issues can properly be handled. Also on layer III clinical analyses can be performed over Clinical Analysis SPARQL Queries (CASQ (3)). On the last layer (IV) a user or a monitoring tool can integratively query distributed clinical data.

Summarizing it, the mapping is performed iteratively in a stack-like manner: The first mapping α(D2R mappings) transforms the relational database layer (I) to the RDF representation layer (II). The next mapping β(N3 and SKOS) transforms the RDF layer II to the Domain Ontology (OO) layer III, where the data is globally queryable as CASQ over mapping γ(DCO and OO).

Advantages and disadvantages: Query complexity and lesser source transparency are known disadvantages of a wrapper-mediator architecture. As a comprising system would need to cover veterinary resistance occurrence, the system can be still seen as limited.

Key advantages:

- The DebugIT monitoring tools access hospital data in real-time, hence allowing early trend discovery and opportunities for early interventions. Most of the other existing monitoring systems have a low timely resolution in reverse.
- DebugIT is not limited to a certain selected set of bacteria or sampling methods, but potentially includes all bacteria and strains as found in the hospital data that can be mapped to a DCO/Uniprot taxonomity.
- DebugIT stores data in a re-usable, semantically formal and traceable way.

[1] http://d2rq.org/
[2] DBLP:books/dp/LeserN2006
[3] http://www.w3.org/TeamSubmission/n3/
[4] http://www.w3.org/TR/2009/REC-skos-reference-20090818/
[5] http://eulersharp.sourceforge.net/

4 Implementation

4.1 Setting up databases

4.2 Wrappers

4.3 Register

4.4 Basic dataspace querying

4.5 Advanced query techniques/features

4.6 GUI

4.7 Validation

5 Glossary

Data source A data source addresses an arbitrary data storage, whose data should be integrated in an information system. ([1, p. 7], own translation)

Integrated or integrating information system A integrated information system is an application, which facilitates the access to different data sources ([1, p. 7], own translation)

Meta data Meta data is data, which describes other data. To decide what is data and what is meta data in a system, is application dependant. Meta data has to be stored, browsed and integrated as well as 'normal' data. ([1, p. 8], own translation)

Bibliography

[1] U. Leser and F. Naumann, *Informationsintegration: Architekturen und Methoden zur Integration verteilter und heterogener Datenquellen.* dpunkt, 2006.

[2] A. Ndjafa, H. Kosch, D. Coquil, and L. Brunie, "Towards a model for multimedia dataspaces," in *Multimedia on the Web (MMWeb), 2011 Workshop on*, pp. 33–37, Sept 2011.

[3] M. Franklin, A. Halevy, and D. Maier, "From databases to dataspaces: A new abstraction for information management," *SIGMOD Rec.*, vol. 34, pp. 27–33, Dec. 2005.

[4] A. Halevy, M. Franklin, and D. Maier, "Principles of dataspace systems," in *Proceedings of the Twenty-fifth ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems*, PODS '06, (New York, NY, USA), pp. 1–9, ACM, 2006.

[5] R. H. S. Abiteboul and V. Vianu, *Foundations of Databases.* Addison-Wesley, 1995.

[6] D. Barbará, H. Garcia-Molina, and D. Porter, "The management of probabilistic data," *IEEE Trans. on Knowl. and Data Eng.*, vol. 4, pp. 487–502, Oct. 1992.

[7] A. H. A. Das Sarma, O. Benjelloun and J. Widom., "Working models for uncertain data," *Proc. of ICDE*, April 2006.

[8] G. Grahne, "Dependency satisfaction in databases with incomplete information," in *Proceedings of the 10th International Conference on Very Large Data Bases*, VLDB '84, (San Francisco, CA, USA), pp. 37–45, Morgan Kaufmann Publishers Inc., 1984.

[9] L. V. S. Lakshmanan, N. Leone, R. Ross, and V. S. Subrahmanian, "Probview: A flexible probabilistic database system," *ACM Trans. Database Syst.*, vol. 22, pp. 419–469, Sept. 1997.

[10] G. Grahne, "Conditional tables," in *Encyclopedia of Database Systems*, pp. 446–447, 2009.

[11] A. K. Chandra and P. M. Merlin, "Optimal implementation of conjunctive queries in relational data bases," in *Proceedings of the Ninth Annual ACM Symposium on Theory of Computing*, STOC '77, (New York, NY, USA), pp. 77–90, ACM, 1977.

[12] I. Elsayed, P. Brezany, and A. Tjoa, "Towards realization of dataspaces," in *Database and Expert Systems Applications, 2006. DEXA '06. 17th International Workshop on*, pp. 266–272, 2006.

[13] S. Agrawal, S. Chaudhuri, and G. Das, "Dbxplorer: a system for keyword-based search over relational databases," in *Data Engineering, 2002. Proceedings. 18th International Conference on*, pp. 5–16, 2002.

[14] L. Guo, F. Shao, C. Botev, and J. Shanmugasundaram, "Xrank: Ranked keyword search over xml documents," in *Proceedings of the 2003 ACM SIGMOD International Conference on Management of Data*, SIGMOD '03, (New York, NY, USA), pp. 16–27, ACM, 2003.

[15] V. Hristidis and Y. Papakonstantinou, "Discover: Keyword search in relational databases," in *Proceedings of the 28th International Conference on Very Large Data Bases*, VLDB '02, pp. 670–681, VLDB Endowment, 2002.

[16] M. Carey, D. Florescu, Z. Ives, Y. Lu, J. Shanmugasundaram, E. Shekita, and S. Subramanian, "Xperanto: Publishing object-relational data as xml," in *In WebDB*, pp. 105–110, 2000.

[17] R. Krishnamurthy, V. Chakaravarthy, R. Kaushik, and J. Naughton, "Recursive xml schemas, recursive xml queries, and relational storage: Xml-to-sql query translation," in *Data Engineering, 2004. Proceedings. 20th International Conference on*, pp. 42–53, March 2004.

[18] Q. He, L. Qiu, G. Zhao, and S. Wang, "Text categorization based on domain ontology.," in *WISE* (X. Zhou, S. Y. W. Su, M. P. Papazoglou, M. E. Orlowska, and K. G. Jeffery, eds.), vol. 3306 of *Lecture Notes in Computer Science*, pp. 319–324, Springer, 2004.

[19] S. H. R. Wurst, *Dataspace Integration in der medizinischen Forschung.* PhD thesis, Technische Universitaet Muenchen, 2010.

[20] "Debugit: Detecting and eliminating bacteria using information technology. a large-scale integrating project." online article.

[21] D. Schober, R. Choquet, K. Depraetere, F. Enders, P. Daumke, M. Jaulent, D. Teodoro, E. Pasche, C. Lovis, and M. Boeker, "Debugit: Ontology-mediated layered data integration for real-time antibiotics resistance surveillance," in *Proceedings of the 7th International Workshop on Semantic Web Applications and Tools for Life Sciences, Berlin, Germany, December 9-11, 2014.*, 2014.

[22] P. Daumke, *Das MorphoSaurus-System - LÃ¶sungen fÃ¼r die linguistischen Herausforderungen des Information Retrievals in der Medizin.* PhD thesis, Universitaet Freiburg, 2007.

[23] R. C. D. T. F. E. C. D. M.-C. J. Ariane AssÃ©lÃ© Kama, Audi Primadhanty, "Data definition ontology for clinical data integration and querying," *Studies in Health Technology and Informatics*, vol. Volume 180: Quality of Life through Quality of Information, pp. 38 – 42, IOS Press 2012.

[24] D. Schober, I. Tudose, and M. Boeker, "Developing dco: The debugit core ontology for antibiotics resistence modelling."

[25] A. D. Sarma, X. L. Dong, and A. Y. Halevy, "Data modeling in dataspace support platforms," in *Conceptual Modeling: Foundations and Applications - Essays in Honor of John Mylopoulos*, pp. 122–138, 2009.

Eidesstattliche Erklärung

Ich erkläre hiermit, dass ich die vorliegende Arbeit selbständig verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel verwendet habe.

Die Arbeit wurde bisher keiner anderen Prüfungsbehörde vorgelegt und auch noch nicht veröffentlicht.

LOCATION, den DATE

David Goeth