University of Passau
Faculty of Computer Science and Mathematics

Bachelor thesis

Implementation of a distributed environment of medical data sources

David Goeth

21. Februar 2018

Bachelor thesis
Chair of Distributed Information Systems
Faculty of Computer Science and Mathematics
University of Passau

Reviewer: Prof. Dr. Harald Kosch
Supervisor: Armelle N. Ndjafa

Abstract

Health care faces the problem of dealing with a vast amount of heterogenous data, that isn't easily accessible. Classical data integration approaches need a global schema before queries on the integrated data can be stated. Dataspace is a new abstraction of data management, that doesn't enforce any schema, allows data-coexistence, and needs only low upfront work for datasources thus being suitable for highly heterogeneous environments. In this thesis we present MeDSpace, a distributed system that allows to state keyword search queries over heterogeneous datasources. The presented system is a test environment for medical datasources and can be used as a starting point for creating a dataspace over multimedia datasources. The system uses medical data, but could be used for any kind of multimedia data that should be searchable by keywords.

Contents

List of Tables

List of Figures

1 Introduction

In the fields of medicine every year a vast amount of (digital) data is produced, that usually is very complex [3, p. 1]. Health data is highly heterogeneous (e.g. different file formats, text data, images, videos) and often lots of tools are necessary to work with the data **TODO: CITE**. In sum, it can be noted that searching data and working with it isn't that easy and in fact, that produces lot's of costs in the healthcare sector **TODO: CITE**.

As a result, the ability to easily access the data would be beneficial for both, research and healthcare institutions [4, p. 2]:

- Lower Costs
- Detecting diseases at early stages
- Simplified collaboration
- Health care fraud detection

The above mentioned benefits were originally stated for Big Data Analysis and Data Mining. For both research fields, as a first step it is important to integrate the data of different datasources into one data integration system. As a first implication it can be noted, that it would be beneficial to create a data integration system for the healthcare sector. This would be an absolute reasonable proposition. But there is another option, the so-called dataspace concept, that doesn't semantically integrate data before it is able to provide its services, and follows a data-coexistence approach. Data co-existence facilitates working with heterogeneous data and many schemas. As heterogeneity is a first-class citizen in a dataspace, it is also useful to integrate multimedia data. That are attractive properties and they form the reason why we will follow the dataspace concept in this work. The dataspace concept was introduced in 2005 as a vision [5]. But to the time of this writing, no dataspace implementation was presented.

In this thesis, we present a distributed environment to state keyword search queries on multiple heterogeneous multimedia medical datasources. The environment could be used as a base for implementing a fully fledged dataspace or for any system, that needs keyword search functionality on multiple (heterogeneous) datasources. Despite the research of dataspaces is very active, they often cover only text data ignoring the properties of multimedia data [6]. This is another reason, why we explicitly chose multimedia data.

It should be noted, that although we use medical data for test data of the system to be presented, the system itself isn't restricted to any kind of particular data.

The content of the thesis is structured as follows: At first we'll go into foundation knowledge of data integration and dataspaces. Then we'll cover related work and reference projects. In Chapter 5 the implementation of our system is presented and

described in detail. At the end, in chapter 6 we will give some outlook, suggestions, and possible improvements to the system.

2 Foundation in Data integration

Data integration constitutes the issue of combining data residing at different sources and providing the user with a unified view of these data [7].

The goal of data 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 [1, p. 3-4].
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 [1, p. 17].
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 [1, p. 60/61].

In general, there are two different types of data 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 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).

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 a result.
Syntactic heterogeneity	includes problems in the presentation of data. This heterogeneity is overcome if all data meaning the same are presented 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 really mean the same by the used names for schema elements. Equal names means consequently equal meaning.

Table 2.1: Kinds of heterogeneity (from [1, p. 60/61], own translation)

2.1 Heterogeneity

From: [1, chapter 3.3 (p.58-78)]

Information systems providing not the same methods, models and structures for accessing their data are called *heterogeneous*. It is often the case, that distributed systems (and therefore are maintained independently) tend to be heterogeneous. In other words, distribution leads often to heterogeneity but not necessarily. De facto, it is to observe, that data sources tend to be the more heterogeneous the more they are autonomous. Two independent systems will in practice always be heterogeneous, even if they contain the same kind of data. Heterogeneity arises from different requirements, different developers and different temporal developments. It even occurs if initially identical software systems are used but over time they were adapted to the specific enterprise's needs. This process is called *customizing*.

Heterogeneity is the main issue of data integration. As a consequence, integrated systems often restrict autonomy of the data sources making them more homogeneous. A common example would be the use of industry-specific standards like common exchange formats, interfaces or communication protocols.

Heterogeneity exists between the data sources as well between the data sources and the integrated system. But often only the latter case is relevant as the data sources often don't communicate among themselves. An example of heterogeneity of data sources and integrated system would be, if the data sources would be provide SQL-access but the integrated would use SPARQL for querying.

It should be clear that in order to overcome heterogeneity, it is necessary to translate queries and the integrated system has to implement functionality, a data source might miss. But this isn't possible in all cases resp. only with big effort. (TODO: Maybe an example?)

As the understanding of heterogeneity is essential for further proceeding, we will go a little more in depth for each kind of heterogeneity:

2.1.1 Technical heterogeneity

With this kind of heterogeneity are meant differences between information systems, that aren't established by their data or descriptions, but by the possibilities accessing the data. There are different technical layers on which heterogeneity can exist:

- **Request function:** query language, parameterized functions, canned queries
- **Query language:** SQL, XQuery, full text search,...
- **Exchange format:** binary data, XML, HTML, tabular,...
- **communication protocol:** HTTP, JDBC, SOAP,...

2.1.2 Syntactic heterogeneity

Syntactic heterogeneity describes differences in the presentation of the same circumstances. For example different number formats (little endian and big endian), different character encoding (Unicode, ASCII) or different seperators in text files (tab delimited and comma separated values (CSV)). Syntactic heterogeneity therefore can be reduced to technical differences in the presentation of information. Whereas the *synonym problem,* which deals with the representation of equal concepts through different names, is an issue of *semantic heterogeneity.*

2.1.3 Data model heterogeneity

Structured information systems describes their managed data using schemas in a certain data model. Thus, data model heterogeneity arises, if the integrated system and a data source manage their data in different data models. Important to note is the fact, that data model heterogeneity is independent from semantic differences. Two systems can use two different data models both describing the same circumstance, e.g. describing data in the object oriented or relational data model. Nevertheless it is to observe that differences in the data model often induces semantic heterogeneity. TODO: Maybe example image of different data model concepts?

2.1.4 Structural heterogeneity

Using the same data model doesn't mean that semantic equal data has to be described equally. We speak of structural heterogeneity, if two schemas are different even though they represent the same extract of the real world (i.d. the intension of their schema elements is equal).

Structural heterogeneity can have many reasons like different preferences of developers, different requirements, using different data models, technical restrictions and so on. This arises from the *design autonomity* of data sources. A Data source has design autonomity if it can freely decide in which way it provides its data, including the data format, the data model, the schema, syntactic represenation of data, the use of key and comprehension systems and the units of values[[1, p.55]].

2.1.5 Schematic heterogeneity

A special case of structural heterogeneity is schematic heterogeneity. Schematic heterogeneity is present if different elements of the data model are used to model the same circumstance. In the relational model it is possible to model information as relations, as attributes or as values, for instance.

Schematic conflicts are particular difficult to resolve, as it isn't usually possible to overcome them with the query language. So is it mandatory to explicitly state attributes and relations in relational languages. If schematic heterogeneous data sources should be integrated, one would define a view. But the query has to be altered if something in these elements is changed.

2.1.6 Semantic heterogeneity

Sole values have no implicit meaning in an information system. Looking at the number '20' on a table doesn't allow conclusions concerning its meaning. In fact, it could be anything. Only through *interpretation* data becomes information, and for interpretation one needs knowledge about the concrete use case and world knowledge. The interpretation of data is also called its *semantic*. To interpret data in a information system, further information is therefore consulted:

- The name of the schema element containing the datum
- The position of the schema element within the schema
- Other data values stored in the schema element

Comprehending all this information, we speak from the data's *context*. Data is obtaining only its meaning by taking into account its context. But it should be considered that some parts of the context are given in machine-readable form (like the schema), and some not (e.g. domain specific knowledge). Also absolute identical schemas having different contexts can have different semantic meanings. Owing to circumstances, an integrated system has to make implicit context knowledge explicit e.g. by introducing new schema elements.

Semantic conflicts relate to the interpretation of names resp. symbols. Semantic conflicts occur in the interaction of names and concepts in different systems. A *concept* means here the intension of a name, the set of real world objects of a name is called its extension. The most common semantic conflicts are synonyms and homonyms. Two names are *synonyms* if they have the same intension, but are syntactic different. Whereas two names are *homonyms* if they are syntactic identical, but their intensions are different. But more difficult to treat are names,

whose intensions neither are identical nor are completely different, but conclude themselves or are overlapping.

In principle, it is difficult to detect semantic conflicts and resolve them clearly. Reasons are that for schema analysis are only available the schema themselves and some example data. Sometimes there is additionally some domain knowledge and documentation of the data sources, but that isn't always the case, particular on autonomous systems. Particular cases occurs always if the concepts aren't clearly defined or are interpreted differently.

On the other hand, complete resolving isn't always possible, only with big effort achievable or simple not necessary. This is also called *remaining blur*.

2.2 Distribution

From: [1, chapter 3.1 (p.51-54)]

An issue of data integration is the distribution of data to be integrated. We speak thereby of data which lies on different systems. It should be noted that it is assumed that the access to the data is ensured, i.d. the systems are connected. A common example of distribution are data sources which can be accessed by a web interface using a browser: The Browser shows only a snippet of the data provided by a web server. The data itself is managed by the web server or one of the headed data sources. Data distribution isn't plainly an annoyance as it is often an intended design decision. Data distribution is handy for load balancing, reliability and protection against data loss. Thereby is the distribution controlled on a central point. The consistency of data is assured using sophisticated mechanisms like the 2-phase.commit protocol. Contrary, in a typical data integration project, distribution of data has historical evolved or is conditioned organizational and is thus uncontrolled redundant.

Distribution can be structured into logical and physical distribution. Data is *physical* distributed, if it lies onto physical different systems which can also geographically be located on different places. In turn, data is *logical* distributed, if there exists multiple possible locations for storing a datum.

2.2.1 Physical distribution

To integrate physical distributed data there have to be overcome several issues: The first step is to detect the physical storage location of the data. Therefore computer, server and port and network have to be identifiable and locatable. For identification, applications used in the internet use Uniform Resource Locators (URLs) to identify remote computers and services not knowing there exact location. The part of locating is the responsibility of the network layer of the computers, using protocols like TCP/IP.

The second problem resulting from physical distribution is data stored in different schemas. Common query languages don't provide the possibility to query tables using different schemas. There are two possible ways to overcome this hurdle:

- Separating a query into several schema specific queries and consolidating the results in the integrated system.

- Developing a language able to handle several schemas in a posed query.

The third problem are changed requirements for the query optimizer: In a central database the optimizer's key function is minimizing the accesses to the secondary storage whereas for a distributed database the accesses over the network should be held as small as possible for this kind of accesses needs considerably more time than accesses to the secondary storage.

2.2.2 Logical distribution

If identical data is located on different places inside the system, we speak of a logical distribution. The key property of logical distribution is therefore the *overlapping of the intension* of different data storage locations. Thereby it is insignificant where these systems physically really are. Two database instances on one computer induce already distribution issues.

A central point here plays the redundancy in a system. A system is called redundant, if semantic equal data can be located at different places. In order to get a consistent view of a redundant system, redundancy has to be strictly controlled (e.g. using triggers or replication mechanisms). At anytime it has to be assured that the same data is at all different places. But data integration typical has to deal with *uncontrolled redundancy* for each data source is maintained independently. This fact results to several problems:

- **Localization**: For a user it isn't obvious in which sources to find specific data. As a result, the integrated system has to provide meta data allowing *to locate data*. This can e.g. done by a catalog of all schemas and their descriptions or a global schema with mappings to source schemas.
- **Duplicates**: If it is possible that data exists on multiple locations, there will exist duplicates, i.d. objects stored on both locations. These objects have to be recognized by the integrated system.
- **Inconsistencies**: Redundant data can contain inconsistencies, which have to be resolved for a homogeneous presentation.

2.3 Architectures

Integrated systems evolved from distributed databases which in turn are based on the classical monolithic database. In the following basic database concepts, on which integrated systems base, are explained. Additionally different types of integrated systems are presented shortly.

The **monolithic database** is subdivided into three layers as seen in 2.1: 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).

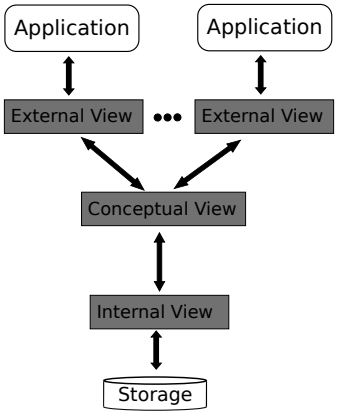

Figure 2.1: Architecture of a monolithic database TODO: own figure but inspired by book; cite properly

Next comes the **distributed database** architectures as shown in 2.2. 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. Dependent on the used distribution strategy, the local conceptual schema is equal to the global conceptual schema or an extract from it. Common distribution strategies are vertical and horizontal partitioning:

- horizontal partitioning: Data of big tables is distributed per tuples on different computers. A union operation consolidates these parts again.
- vertical partitioning: Data of big tables is distributed per attributes on different computers. Each partition contains additionally a shared key attribute. This makes it possible to consolidate the partitions with a join operation.

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. They are strictly checked while conception and operation and thus the main problems of heterogeneity (like structural and semantic heterogeneity) don't occur ([1, p. 91-93], own translation).

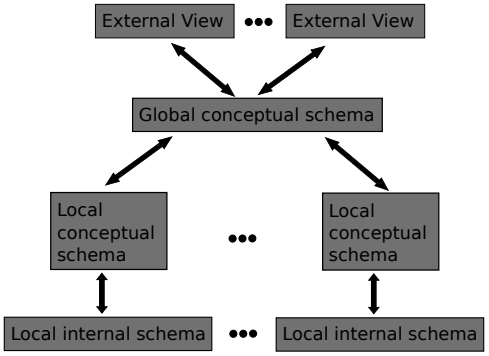

Figure 2.2: Architecture of a distributed database

For allowing to connect heterogeneous databases, **Multidatabase Systems** (MBS) have been evolved (shown in 2.3). 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 language. 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).

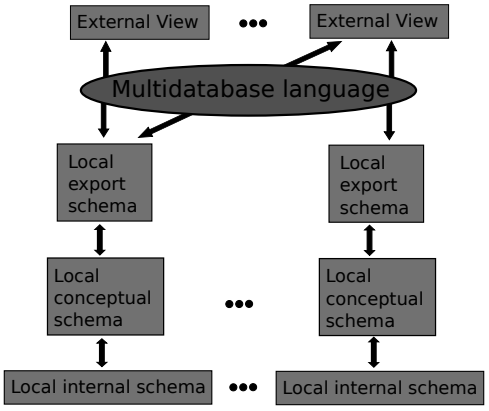

Figure 2.3: Architecture of a multidatabase system

In contrast to MBS, **federated database management systems** (FDBMS) have a global conceptual schema as seen in 2.4. 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 an 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 and the local export schema. This new layer is called *local component schema* and translates the local conceptual schema into the canonical data model. The translation is done by a software component called *Wrapper*. All in all, a 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).

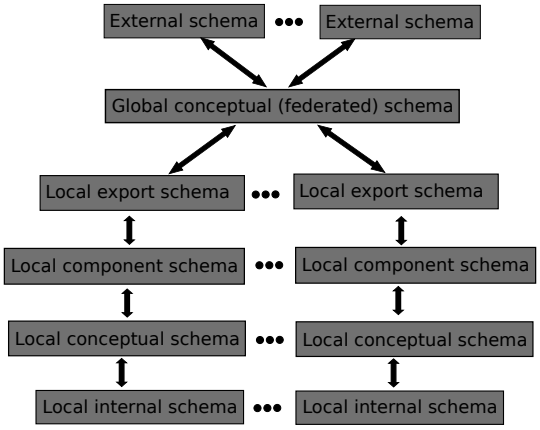

Figure 2.4: Architecture of a federated database management systems (FDBMS)

Mediator-based information systems (as seen in 2.5) are a generalization of the previous architectures, as they know only two separate components, namely Wrappers and Mediators: Wrappers 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. It realizes the communication between the mediators and data sources. To its task count:

- overcoming interface heterogeneity (e.g. SQL to html formulars)
- overcoming language heterogeneity. This includes the handling of restricted data sources and different query languages.
- Providing data model transparency through translating the data into the canonical data model.
- resolving schema heterogeneity by using a suitable mapping between the source schema and the global schema.
- Supporting the global query optimizing by providing information about the query capabilities of the data source and expected costs.

As the system to be created within this thesis uses wrappers, we go more in depth and look at the design, architecture and implementation goals of an architecture using wrappers. The following statements are results of from [8]:

Low start-up costs: Writing a wrapper should be possible to do very quickly, very simple wrappers even in matters of hours. The authoring of a wrapper should

also be as simple as possible, so that the wrapper can be written with little or no knowledge about the internal structure of the integrated system.

Easy evolving: Evolving should be done very easy due to two reasons: A wrapper should be implemented very fast to show feasibility. More sophisticated features a data source can provide, should be added later. Additionally can the data source change itself over time and the wrapper should be easily adaptable.

On the other hand are mediators software components, which use knowledge of certain data to *create and provide* information for other applications. In a mediator-based architecture, mediators access one or more wrappers and deliver a specific value, normally structural and semantic data integration.

The following list shows an extract of the services, a mediator might provide (from, [9, p. 5-6]):

- Selection of likely relevant material
- Invocation of wrappers to deal with legacy sources
- Resolution of domain term terminology and ontology differences
- Sending the information and meta-data to the customer application
- Imposition of security filters to guard private data
- Omission of replicated information
- Assessment of quality of material from diverse sources
- Integration of material from diverse source domains based on join keys

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).

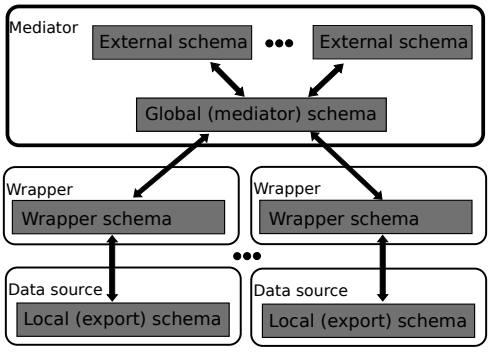

Figure 2.5: Architecture of a mediator-based information system

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).

3 Dataspaces

The concept of dataspaces was firstly presented in [5] and describes a new abstraction of data management. 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 meta-data. 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 an 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.

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.

3.1 Components and Services

Used/discussed Papers: [10], [11]

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.

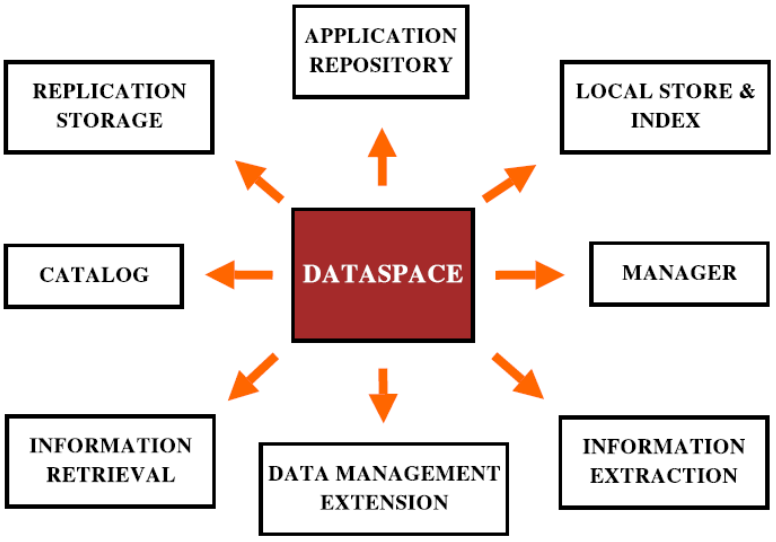

Figure 3.1: Environment of a Dataspace

Catalog: 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 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.The catalog 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. Then, it can support basic browsing over the participant's inventories. The catalog may reference a meta-data repository to separate the basic and more detailed descriptions. 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.

Information Retrieval: The Information Retrieval component (in [11] stated as 'Search and Query') can be split up into queries and searches, two different methods of information retrieval and represent together one of the main services a DSSP 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 any 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 (do more research, discuss in more detail; Why is it challenging?). The development of

methods for keyword searching on relational and XML databases was done by the data-engineering community [12, 13, 14], 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 [15, 16]. In the following are listed the main requirements having to be supported by this component:

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.

Structured queries 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.

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) time steps 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.

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: 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. The component should try to achieve the following goals:

- to create efficiently queryable associations between data items in different participants. Important is here, that the index 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 component (not listed in Figure 3.1) 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.

Data Management Extensions: This component (in [11] stated as *Source Extension 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 DSSP to provide additional functionality such as backup, recovery and replication. 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.

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 [17]. 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.

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.

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.

3.2 Query answering

Used paper: [11]

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.

3.3 Uncertainty in Dataspaces

In the paper "Data Modeling in Dataspace Support Systems" [18], 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. 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.

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 [12], [14]) 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.

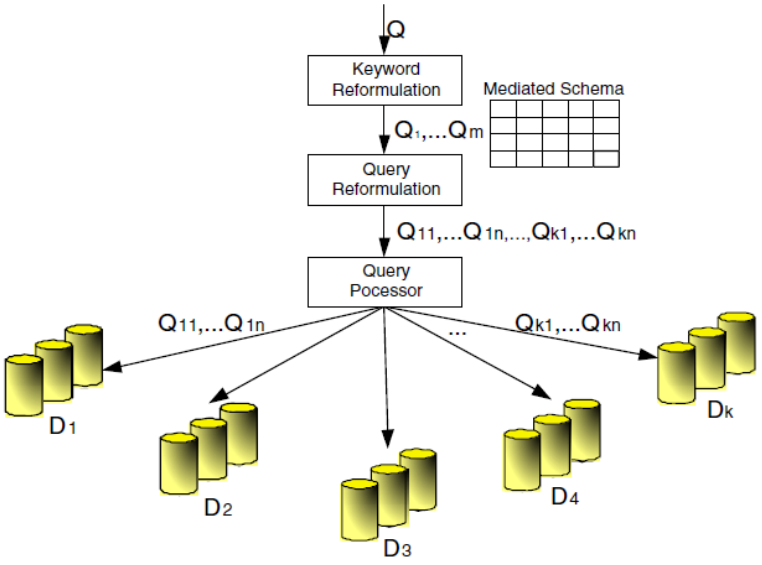

Figure 3.2: Architecture of a data integration system that handles uncertainty
TODO: Discuss image more in depth?

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 3.2. 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.

3.3.1 Probabilistic Mediated Schema Mapping

TODO
Used Papers: [19]

In [19] was shown, that it is possible to automatically bootstrap a data integration system that answers queries with high precision and recall. In a sense, it is possible to set up a data integration system without any human involvement. With this approach, it isn't possible to answer queries fully accurate and complete as it would be with manual or semi-automatic ones, but best-effort answers and improvement over time are possible by using a probabilistic data model. Mediated schemas can be build automatically by clustering attributes from the various source schemas to groups by their semantic meaning and then using this groups as attributes in the mediated schema. Sources in a dataspace typically are heterogeneous, so as a general rule there is more than one possibility to cluster the attributes and it exists an uncertainty of the best(s) grouping(s). If you would choose only one schema of the set of possible mediated schemas, you would likely have a loss in query answering in both terms, precision and recall. So it seems natural to consider all possible mediated schemas combined in a single mediated schema. By weighting each mediated schema through its likelihood of accuracy, it is possible to favor schemas with a greater weight for query answering and answer ranking. And that is exactly, what a probabilistic mediated schema (p-mediated schema) is doing. For clustering attributes from the source schemas, similarity functions based on attribute matching are used. Thus, some not very obvious attribute correspondences aren't detected by this approach. But the authors believe, that using more advanced schema matching algorithms like also considering column value similarities, could minimize this problem [19].

The concept of probabilistic Schema Mapping (p-mapping) was introduced in Cite!!!. It describes a probabilistic distribution of possible mappings between a source and a mediated schema. A mathematical formal definition and additional information about probabilistic schema mapping can be found in the original paper [19].

The whole process is visually represented in 3.3, which shows the architecture of the data integration system used by [19].

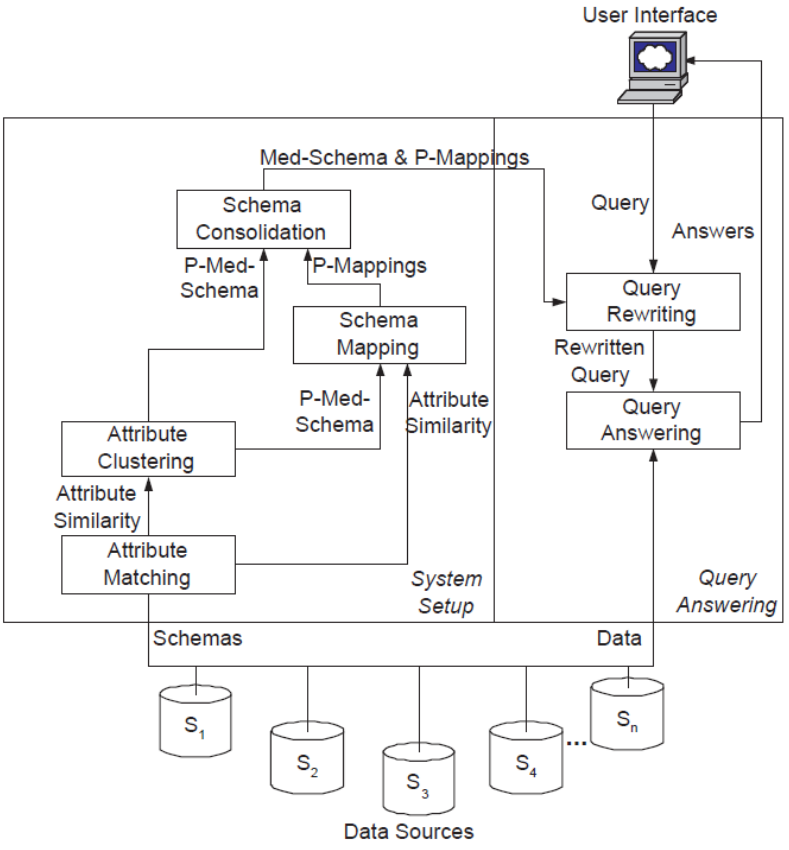

Figure 3.3: Bootstaping a 'Pay as you go' data integration system

On set-up time, the system automatically generates the p-mediated schema and appertaining p-mappings by clustering the attributes of the source schemas and calculating mapping probabilities afterwards. For the clustering process similarity functions are used, but more advanced techniques could used, as well. After creating the p-mediated schema and its p-mappings, they are consolidated to generate a final mediated schema and p-mappings. Consolidation means to aggregate all possible schemas to one mediated schema and creating p-mappings for this single schema. The purpose of the consolidation is providing the user a sole mediated schema to interact with and additionally it speeds up query answering, as the consolidation process removes redundant information from the p-mediated schema and therefore queries only need to be rewritten and answered based on one mediated schema. At query-answering time the query is rewritten for each data source according to the mappings and answers the rewritten query on the data sources. Answering queries with respect to p-mappings returns a set of answer tuples, each with a probability indicating the likelihood that the tuple occurs as an answer.

As probabilistic schema mapping is crucial for dataspace systems we want to go a little more in depth by showing a little example of bootstrapping a fictional data integration system providing access to bio-medical information in the field of antimicrobial susceptibility tests:

Let us assume we have 2 data sources S_1 and S_2 . Their concepts and data sets are shown in 3.4.

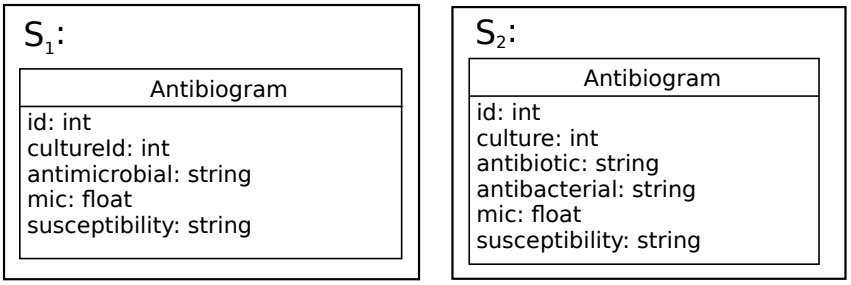

Figure 3.4: Concepts of our example

Beside syntactic heterogeneity, the attribute members *antibiotic* and *antibacterial* from S_2 could both possibly matched to *antimicrobial* from S_1. Clustering using similarity function could thereby result in the following two clusterings (other possibilities are omitted for the sake of simplicity):

M_1: ({*id*}, {*cultureId*, culture}, *drug* := {*antimicrobial*, antibiotic}, {antibacterial}, {mic}, {susceptibility})
M_2: ({*id*}, {*cultureId*, culture}, *drug* := {*antimicrobial*, antibacterial}, {antibiotic}, {mic}, {susceptibility})

To break it down, the two clusterings show whether *antibiotic* or *antibacterial* are semantically equal to *antimicrobial*. Let us assume that both schemas are equally probable, this means $P(M_1) = P(M_2) = \frac{1}{2}$.
Consequently the p-mediated schema is given by:
$M_{\mathrm{P}} : \{(M_1, \frac{1}{2}), (M_2, \frac{1}{2})\}$
The appertaining p-mappings are given by 3.1 (note, that the probabilities are only for demonstration purposes and weren't calculated):

Possible Mapping for M_1	**Probability**
m_{11} := {(id, id), (culture, cultureId), (antibiotic, *drug*), (antibacterial, antibacterial), (mic, mic), (susceptibility, susceptibility)}	$\frac{9}{10}$
m_{12} := {(id, id), (culture, cultureId), (antibacterial, *drug*), (antibiotic, antibacterial), (mic, mic), (susceptibility, susceptibility)}	$\frac{1}{10}$
Possible Mapping for M_2	**Probability**
m_{21} := {(id, id), (culture, cultureId), (antibacterial, *drug*), (antibiotic, antibiotic), (mic, mic), (susceptibility, susceptibility)}	$\frac{8}{10}$
m_{22} := {(id, id), (culture, cultureId), (antibiotic, *drug*), (antibacterial, antibiotic), (mic, mic), (susceptibility, susceptibility)}	$\frac{2}{10}$

Table 3.1: Probabilities of the p-mappings

Now, let's assume, we have the following data set in S_2 containing only one tuple: {(3180102, 1910181, cefepime, axepim, 0.96, S)}; And assume we want to process the query:

Select drug, mic, susceptibility
from Antibiogram

Using the schemas M_1 and M_2, the possible answers would be: a_1 := (cefepime, 0.96, S)

a_2 := (axepim, 0.96, S)
To calculate their probabilities and therefore getting the order for a Top-k ranking, we have to consider both, the probability of the schemas and their attending p-mappings. Therefore:
$P(a_1) = P(M_1) \cdot P(m_{11}) + P(M_2) \cdot P(m_{22}) = \frac{1}{2} \cdot \frac{9}{10} + \frac{1}{2} \cdot \frac{2}{10} = \frac{11}{20}$
$P(a_2) = P(M_1) \cdot P(m_{12}) + P(M_2) \cdot P(m_{21}) = \frac{1}{2} \cdot \frac{1}{10} + \frac{1}{2} \cdot \frac{8}{10} = \frac{9}{20}$

Finally, the top-k answer looks like:

Answer	**Rank**
(cefepime, 0.96, S)	0.55
(axepim, 0.96, S)	0.45

Table 3.2: Query answer of our example

3.4 Towards a Model for Multimedia Dataspaces

In the paper 'Towards a Model for Multimedia Dataspaces'[6] 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.

Important 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,...).

4 Reference Projects

4.1 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 [20, 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[21]. The team of the DebugIT project released the architecture of their system in [22]. 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 4.1 shows the layered architecture of the DebugIT's system:

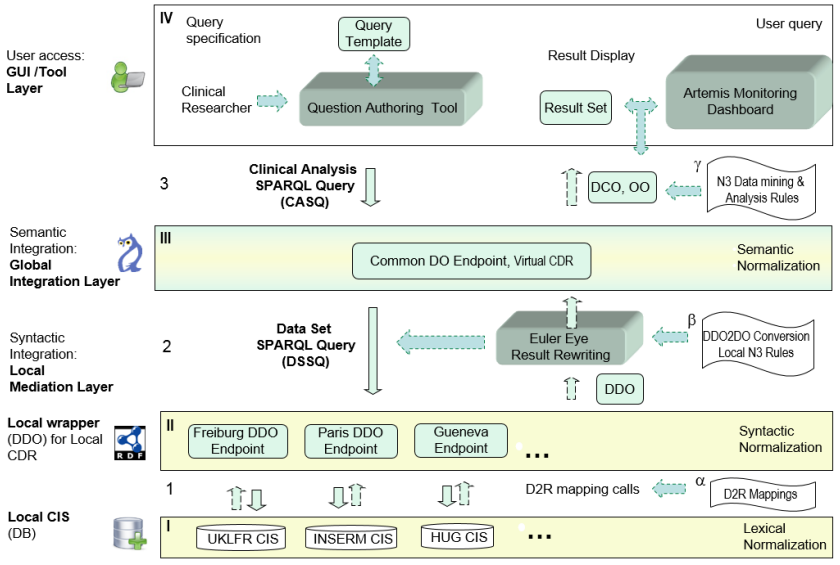

Figure 4.1: The layered mediator architecture of the DebugIT project

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[23]. 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) [24]. 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) [25], 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/

5 Implementation

Importance of unicode

A dataspace includes data sources possibly spread around the globe. It is near inferring to support a wide area of different languages. In terms of character sets, it is therefore necessary to use a unicode encoding. Unicode is a system assigning each character a unique code point and it is designed to support the worldwide interchange, processing and display of texts written in different languages[26]. There exist several encodings for unicode. The more well known are the UTF and UCS encoding families. In the draft of HTML5 it is advised to use UTF-8 for new web pages[27]. Thus, to simplify processing, we follow the recommendation and use UTF-8 throughout the dataspace. If a data source doesn't use UTF-8, it is the task of its wrapper to do a proper conversion to UTF-8.

5.1 Setting up databases

5.1.1 Setting up a MySQL data source

To setup a MySQL data source download a recent stable MySQL community server[6] and install it for you target platform. Additionally you will need the Connector/J components, the official JDBC driver for MySQL.

At time of writing, the most recent stable versions are the community server 5.7.17 and the Connector/J 5.1.41. These versions are used for the thesis project and all following commands are related on them. If you're using different versions, assure that the instructions are adapted properly. As a detailed installation instruction for all supported platforms would break the mold, the reader is encouraged to consult the official manual[7]. Assure that the MySQL binary folder is integrated into your class path, so that you can access it globally in a shell/command line. Although not necessary it is recommended for security reasons to set a password for the root user[8]. After installing the server, do postinstallation setup and testing[9].

To support UTF-8, set in your my.cnf

```
default-character-set = utf8
```

in the mysql section and

[6] https://www.mysql.de/downloads/
[7] https://dev.mysql.com/doc/
[8] https://dev.mysql.com/doc/refman/5.7/en/default-privileges.html , https://dev.mysql.com/doc/refman/5.7/en/resetting-permissions.html
[9] https://dev.mysql.com/doc/refman/5.7/en/postinstallation.html

```
character-set-server=utf8
collationserver=utf8_general_ci
```

in the mysqld section. Then restart the mysqld daemon. In the following, it is assumed, that you have a running MySQL server now that can be accessed via shell/command line. Before you connect to the MySQL server, you should assure that the application you use for connecting is using UTF-8 for user input and sending statements. So, validate that your shell/command line is using UTF-8. E.g. on windows system the command line isn't using UTF-8 by default [10]. Now try to connect to the database as the user root:

```
mysql -u root -p
```

If you've done all right, you should be connected to the database after entering and confirming the password, that you've previously stated for the user root.

The next step is to validate, that UTF-8 is indeed continuously used. Execute:

```
SHOW VARIABLES LIKE 'char%';
```

and check, that the variables *character_set_client*, *character_set_connection*, *character_set_database*, *character_set_results*, *character_set_server* and *character_set_system* are all set to utf8. Basically, these variables are used to interpret and write data consistently in UTF-8. More information about the stated variables can be found on the manual [11].

The next step is to initialize the data source with a database and some content. Further we need a user which is used by the wrapper to communicate with the data source. The wrapper needs no writing rights and indeed we don't want it to change the data, so following the security rule 'As few rights as possible' we grant that user only reading rights for fetching data. The commands for initializing the data source and creating a read-user are in the file **init_mysql.sql** which is located in the appendix data in the folder **implementation/SQL/mysql**.

To execute commands from a file, execute while logged in as the root user:

```
SOURCE path_to_sql_file;
```

where *path_to_sql_file* is the full (absolute or relative) path to the sql file. Per default, the created database will be named **medspace** and the read-user will be called **medspace_client**. If you want edit the init.sql file, beware that the file is encoded in UTF-8. As we instructed mysql to use UTF-8 in every case, this is encoding is required. Assure that your file editor saves the file in that encoding, too.

[10] To set the encoding to UTF-8 on the windows command line, change the active code page to 65001 and set 'Lucida Console' as the displaying font. In contrast to the font, the code page is only active for the current console session. But you can automate this command with a AutoRun setting. For more information see `https://blogs.msdn.microsoft.com/oldnewthing/20071121-00/?p=24433`

[11] `https://dev.mysql.com/doc/refman/5.7/en/server-system-variables.html#sysvar_character_set_client`

5.2 Wrappers

As described in Chapter 3, a wrapper is an interface between the datasource and the dataspace. The wrapper can provide any number of services to access the data of the datasource, but the one service, that every wrapper has to implement, is the keyword search. In the thesis' project there are three datasources: A SQL database, a pdf file server and a SQL multimedia database serving image files. The SQL database and the multimedia image database contain structured data while the pdf file server contains semi-structured data. The following sub sections describe the functionality and implementation of the wrappers for the three datasources in detail.

Keyword query language convention
As already previously told, the one service that every wrapper has to implement is the keyword search functionality. In order to search for specific keywords, it is important to define a convention how the keyword search should work, since there are several possibilities. E.g. if a query is posed with two keywords, should a query be created for all results that contains both keywords or it is enough if one of the keywords is contained? Should boolean operators be allowed (AND, OR, NOT)?

The design decision is at follows: Rudimentary keyword search functionality suffices and several keyword searches have to be interpreted in an 'OR like' manner, i.d. one of the stated keywords suffices to fulfill the query condition. The reasons are, that no complex global query language is necessary, that all wrappers have to implement which leads to much more flexible implementation decisions. As wrappers can provide as many services as they like it is easy to provide a much more powerful keyword search service. So this design decision doesn't restrict the power of a wrapper. So, why is an 'OR like' interpretation used for a multi-keyword statement and not an 'AND like' one ? There is no special reason for that design decision, as both possibilities would be conceivable, since in the dataspace services could be implemented, that would complement the missing feature:
A dataspace is supposed allowing to refine iteratively a keyword search result. Assuming that keywords are ORed, this condition allows to implement a service that allows the user to search for one keyword and than for another on the result set. The result would be a executed AND operator. Similarly could a service be implemented that would add an OR operator.

5.2.1 SQL Wrapper

The task of the SQL Wrapper is to convert the SQL data into RDF and providing a keyword search functionality, as SQL databases doesn't provide such a functionality.
In the thesis' project a MySQL datasource is used. But as several other SQL database vendors exists, one additional design goal is to produce a wrapper that can be used with any other SQL database. That will considerably reduce prospective maintenance work.
Thus, a SQL wrapper was implemented, that doesn't use any vendor specific features.

At first, we want to look at the conversion from SQL to RDF. To do this, the wrapper implements a specialized version of the D2rMap language. D2rMap was designed by Chris Bizer and is a declarative language to describe mappings between relational databases schemata and OWL/RDFS ontologies[2].

D2rMap is a general purpose language to export any SQL data to RDF. To better suit the needs for a dataspace wrapper, the language was changed. The custom language is called MeDSpace D2rMap and its language specification can be found in the appendix.

The mapping is done as follows: At first the user specifies mappings in a config file. Each mapping is used to create RDF instances of a certain type. The mapping contains a SQL query, that represents all the data, that is necessary to create the instances. Furthermore, in the mapping are columns specified, that are used to create unique IDs for the created RDF instances. The next step is to fetch the SQL data and group the record set according to the fore mentioned columns. Now, each row of the grouped record set represents a RDF instance, so the instances can be created by proceeding all rows. The last step is the creation of the property statements of the RDF instances. Important to note is the seperation of the last two steps. As all RDF instances exists before the properties are created, it is possible to reference other RDF instances (from the same mapping or another). The mapping process is visualized in figure 5.1.

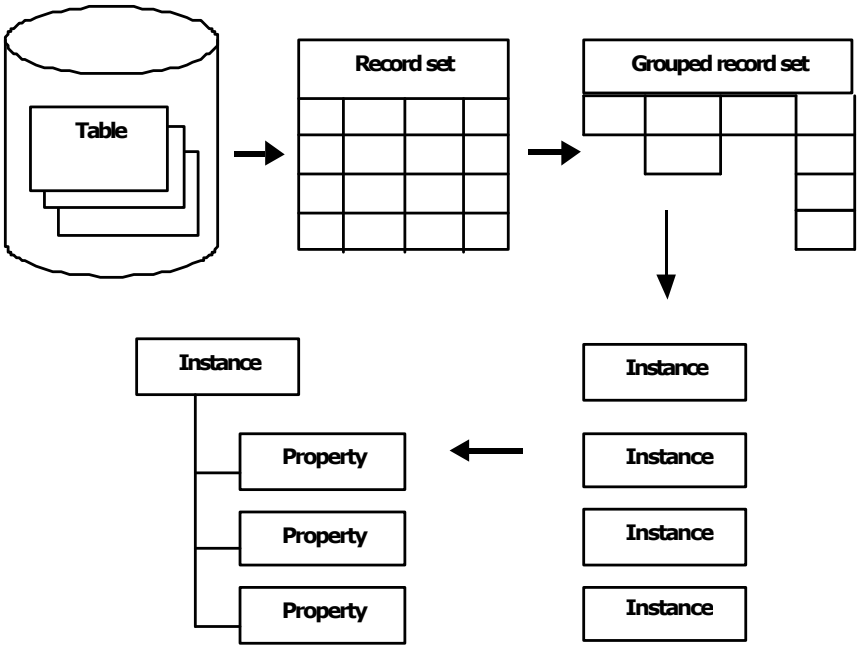

Figure 5.1: The D2r mapping process; Taken from [2]

After discussing the SQL to RDF mapping, we look at the keyword search, now: Mainly there are two possibilities, to implement a keyword search functionality:

- Construct a keyword search query in SQL and let the database answer the query.
- Use a keyword search engine that answers the query based on an external index

At first glance, the first option sounds obviously simple, but after implementing it showed several disadvantages: SQL is not designed to provide search functionality based on keywords. SQL uses the ***LIKE*** operator for pattern matching. But in order to do a Full-text search, the SQL Query executor cannot use any index resulting in poor query answer performance. Another problem of *LIKE* is, that there is no way to define, that only whole words should be searched and not just sub word matching. Whole word matching is very important, as e.g. a user searching for data about male patients should not also get data about female patients. As a result, the *LIKE* operator is not suitable for a proper keyword search service as expected to be provided by a dataspace wrapper. Several SQL database vendors provide often own solutions for Full-Text search queries. But these solutions have often other restrictions as e.g. only column fields having the datatype *TEXT* (on MySQL) and the fields have to be specified as fulltext fields (in their creation or through an update operation), so that the SQL engine is able to create an index for it (at MySQL databases at least).
A Wrapper could use vendor specific services but that would exclude other SQL database vendors, obviously.

The second option doesn't rise the aforementioned issues of option one. For the Wrapper a keyword searcher was implemented using the fulltext search engine Apache Lucene Core [12]. The advantage of using Lucene is it's high-performance and scaling of keyword searches over large data sets. Additionally it allows a fine granular configuration about the query construction and sorts automatically the query result by relevance (so called query result ranking).
The major disadvantage of using lucene is that the SQL data have to be extracted and indexed outside the database. If the data changes or rows are added resp. deleted, the index has to be updated accordingly. The update process can be very complex, as not only new data has to be indexed resp. existing data has to be removed, but also data that references the deleted or new data that depends on it. A simpler but obviously slower solution is to reindex the whole data set. Reindexing the whole data set is only advised if updates occur not that often or if it is acceptable if the wrapper updates the index not instantly and thus provides potentially outdated data.

The decision which method is more suitable depends primarily on the use case and the domain. As the project is designed to be used as a test suite for medical datasources and medical science, it is acceptable if the data is outdated to some degree and will be updated not very frequent. So, changes on the datasource haven't to be updated in near-realtime.
Than, one of the design goals for this project is to design a SQL wrapper that can be used with arbitrary vendors. As a result, vendor specific services are not an option, too.
Having this in mind, the preferred method for the keyword search functionality clearly is Apache Lucene Core, as the advantages clearly outweigh its disadvantages. Thus, a full functional keyword searcher was implemented powered by Lucene.

TODO: keyword search services: The basic service and the advanced one, that utilizes the whole potential of Apache Lucene

[12] `https://lucene.apache.org/core/`

TODO: Communication with the register

5.2.2 Image Wrapper

fgfg

5.2.3 PDF Wrapper

5.3 Register

5.4 Basic dataspace querying

5.5 Advanced query techniques/features

5.6 GUI

5.7 Validation

6 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 An 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 dependent. Meta data has to be stored, browsed and integrated as well as 'normal' data. ([1, p. 8], own translation)

Schema The word comes from greek $\sigma\chi\acute{\eta}\mu a$(skhēma) meaning *shape* or *plan*. In computer science the word has different meanings. In the context of database theory the word is used for a representation of the structure (syntax), semantics, and constraints on the use of a database (or its portion) in a particular data model. [28, p. 235]. And an XML schema defines the structure, content and semantics of XML documents[29]. To generalize it, a schema in the context of data integration is the definition of the structure, content and semantics of a data source.

Bibliography

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

[2] C. Bizer, "D2r map – a database to rdf mapping language," 01 2003.

[3] D. Tomar, "A survey on data mining approaches for healthcare," vol. 5, pp. 241–266, 10 2013.

[4] W. Raghupathi and V. Raghupathi, "Big data analytics in healthcare: promise and potential," *Health Information Science and Systems*, vol. 2, p. 3, Feb 2014.

[5] 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.

[6] 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.

[7] M. Lenzerini, "Data integration: A theoretical perspective," in *Proceedings of the Twenty-first ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems*, PODS '02, (New York, NY, USA), pp. 233–246, ACM, 2002.

[8] M. T. Roth and P. M. Schwarz, "Don't scrap it, wrap it! a wrapper architecture for legacy data sources," in *Proceedings of the 23rd International Conference on Very Large Data Bases*, VLDB '97, (San Francisco, CA, USA), pp. 266–275, Morgan Kaufmann Publishers Inc., 1997.

[9] G. Wiederhold, M. Genesereth, and M. Papazoglou, "The conceptual basis for mediation services mail to issue editor the conceptual basis for mediation services," 1996.

[10] 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.

[11] 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.

[12] 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.

[13] 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.

[14] 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.

[15] 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.

[16] 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.

[17] 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.

[18] 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.

[19] A. Das Sarma, X. Dong, and A. Halevy, "Bootstrapping pay-as-you-go data integration systems," in *Proceedings of the 2008 ACM SIGMOD International Conference on Management of Data*, SIGMOD '08, (New York, NY, USA), pp. 861–874, ACM, 2008.

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

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

[22] 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.

[23] P. Daumke, *Das MorphoSaurus-System - Lösungen für die linguistischen Herausforderungen des Information Retrievals in der Medizin.* PhD thesis, Universitaet Freiburg, 2007.

[24] R. C. D. T. F. E. C. D. M.-C. J. Ariane Assélé 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.

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

[26] "About the unicode® standard."
`http://unicode.org/standard/standard.html`. Accessed: 2017-04-07.

[27] "Html5 a vocabulary and associated apis for html and xhtml w3c recommendation 28 october 2014."
`https://www.w3.org/TR/html5/document-metadata.html#charset`.
Accessed: 2017-04-07.

[28] A. P. Sheth and J. A. Larson, "Federated database systems for managing distributed, heterogeneous, and autonomous databases," *ACM Comput. Surv.*, vol. 22, pp. 183–236, Sept. 1990.

[29] H. T. C. M. Sperberg-McQueen, "Xml schema."
`https://www.w3.org/XML/Schema`, 2000. Last accessed on 02/15/2018.

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