TECHNISCHE UNIVERSITÄT MÜNCHEN

Fakultät für Informatik

Lehrstuhl III - Datenbanksysteme

Incremental Ontology-Based Integration for Translational Medical Research

Diplom-Informatiker Univ.

Fabian Praßer

Vollständiger Abdruck der von der Fakultät für Informatik der Technischen Universität München zur Erlangung des akademischen Grades eines

Doktors der Naturwissenschaften (Dr. rer. nat.)

genehmigten Dissertation.

Vorsitzende:	Univ.-Prof. Dr. C. Eckert
Prüfer der Dissertation:	
1.	Univ.-Prof. A. Kemper, Ph.D.
2.	Univ.-Prof. Dr. K. A. Kuhn

Die Dissertation wurde am 24.10.2012 bei der Technische Universität München eingereicht und durch die Fakultät für Informatik am 06.03.2013 angenommen.

Abstract

Translational medical research is an emerging concept that aims at transforming discoveries from basic sciences into diagnostic and therapeutic applications. In the opposite direction, clinical data are needed for feedback and as stimuli for the generation of new research hypotheses. This process is highly data-intensive and centered around the idea of integrating data from basic biomedical sciences, clinical sciences and patient care. Therefore collaboration and information exchange is needed between previously separated domains, many of which are themselves fragmented. The complexity and heterogeneity of the involved data is constantly growing with increasing scientific progress and related biomedical structures and processes are subject to rapid change. For this reason, structured domain knowledge, e.g., from knowledge bases, is often required in order to adequately understand and interpret results. Furthermore, integration solutions have to be robust and flexible enough to handle changes in data and metadata. Security and privacy aspects are highly relevant and require the incorporation of complex access control mechanisms as well as concepts for data anonymization and pseudonymization.

In this thesis, firstly an ontology-based methodology for integrating heterogeneous biomedical datasets in a distributed environment is proposed. It advocates an incremental approach that builds upon data coexistence and aims at carrying out semantic integration in a demand oriented and flexible manner. The importance of structured domain knowledge is addressed by blurring the boundaries between primary data and metadata. Data federation allows researchers to maintain control over their local datasets and is also utilized to model a fine-grained access control mechanism. Robustness is achieved by designing the system as a set of loosely coupled components, which can be added, altered and removed independently of each other.

Secondly, an implementation based on a large distributed graph of uniquely identifiable nodes is presented. The groundwork is laid by novel techniques for mapping biomedical data sources into the graph. As all further components require an integrated access to the global data, several compile-time and and run-time optimization techniques for the efficient distributed execution of queries are presented. Manual semantic integration is supported by concepts for browsing, annotating and mapping data items. Automated semantic integration and data transformation is supported via a flexible workflow engine, which builds upon the querying interface. Here, result sets can be post-processed with a scripting language that provides domain-specific operators, such as semantic reasoning and data anonymization. The resulting transformed data can be re-integrated into the graph.

Finally, a prototypical implementation is presented which integrates the individual components into a comprehensive data integration solution. It provides a querying interface for applications and allows to administer the data space via a unified graphical user interface for data integrators.

Contents

List of Figures

List of Tables

CHAPTER 1

Introduction[1]

Scientific discoveries in medicine often begin at „the bench", i.e. in the laboratory, where foundational scientists study disease at a molecular or cellular level. Their discoveries need to be transformed into diagnostic and therapeutic applications at the patients' „bedside" [NIH]. This process is complex and time-consuming, comprising successful tests in clinical studies. The literature usually identifies two „Translational Gaps" between the different areas involved: from basic biomedical research to clinical research and from clinical research to (community) practice [Zer05]. In the opposite direction, clinical observations are needed at the bench. The former director of the U.S. National Institutes of Health (NIH) has characterized this by stating: „At no other time has the need for a robust, bidirectional information flow between basic and translational scientists been so necessary“ [Zer05]. While basic researchers provide the clinicians with new knowledge for treating their patients, they need a clinical assessment of their suggestions. Clinical trials and epidemiological registries (population and patient oriented disease-specific data collections [Fei01, Van08]) are substantial examples for this feedback. Moreover, trials and registries are valuable for the generation of hypotheses from observations. In this field, biobanks (storage facilities for biomaterial samples with annotating data) play an important role, as analyses of biosamples provide insights into cellular mechanisms at fine grained levels including genetic information. Improving the connection between observations and findings from basic and clinical research not only accelerates the translational process but also helps to understand complex multifactorial correlations and close numerous knowledge gaps [Woo08]. As the described processes are highly complex and have to be applied at a broad scale, information technology is one of the key enablers for translational medical research [PES09, Sar10, But08].

1.1 Problem statement

Translational medical research aims at integrating *diverse scientific areas* (biochemistry, molecular biology, medicine, statistics, epidemiology and others). An example for such an interdisciplinary research process is network medicine. The basic idea is to investigate disease mechanisms by building and correlating networks. As is shown in Figure 1.1, cellular components are related by regulatory, protein-protein and metabolic interactions. On a social level, individuals are, e.g., related by their social interconnections. The diseases themselves can be organized in a network according to similar

[1]Parts of the work presented in this chapter appeared in [PWL$^+$11b] and [PWL$^+$11a]

genetic origins. The correlations between these individual networks give new insights into the classification and understanding of diseases [Bar07].

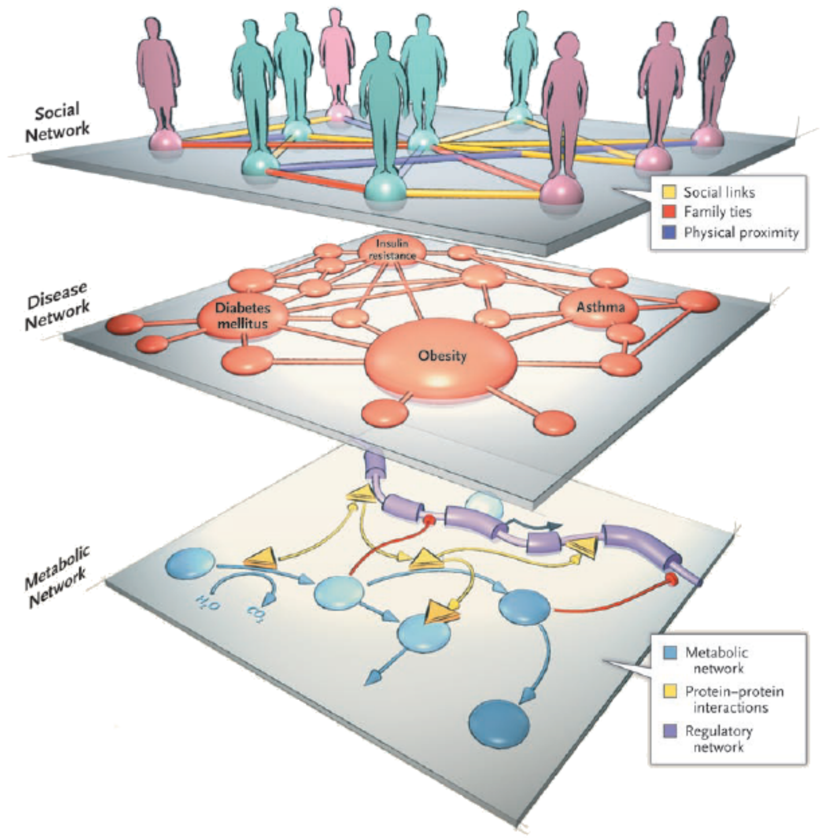


Figure 1.1: Network medicine [Bar07]

Already this example shows that translational research is highly *data intensive*; it requires patient-centric phenotype data (electronic patient records, data from clinical studies) as well as information from molecular biology. These data need to be linked to each other, which requires highly *complex integration efforts*. Furthermore, the complexity and heterogeneity of relevant data is constantly growing with increasing scientific process. Modern datasets include up to millions of different variables. Analytical methods like genomic, transcriptomic, proteomic or metabolomic analyses („omics“) produce large amounts of data, which are stored in data or knowledge bases throughout the world (e.g. [UNI,KEG,OMI]). At the same time trials or registries also continuously generate data at *high volumes*. This leads to a situation in which neither a single researcher nor a group of researchers is able to possess enough domain knowledge to adequately understand and interpret results. The required contextual knowledge is distributed among diverse sources, such as public knowledge bases, terminologies, ontologies or scientific publications [PES09]. Typical use cases for integration solutions include the linkage and mapping of phenotypes and genotypes from research or clinical systems, IT support for clinical trials, the provision of data for statistical analyses and the integration of knowledge bases [EKP09]. An integration solution therefore needs to not only integrate data from *patient care*, *clinical research* and *basic sciences*, but also comprehensive *metadata* and *domain knowledge*.

There are several further domain-specific challenges such as data privacy, security and the fact that relevant data is complex, distributed, heterogeneous and constantly changing. For example, information about biomolecular pathways often resembles *graph structures*, which lead to unique challenges for data management. The separation of domains (i.e., health-care, clinical science, basic sciences), which are themselves fragmented (e.g., into practitioners, clinical care, ambulatory care), has lead to a landscape of *distributed*, *autonomous* information systems managing very *heterogeneous* datasets. In addition, legacy-systems and ad-hoc solutions without adequate integration concepts are in widespread use [AAT08]. To a restricted extent, well-understood integration architectures, such as data warehouses or federated solutions are being deployed to provide integrated views on distributed sources of information. Standards for the exchange of medical data exist (e.g. HL7 and DICOM), but inter-institutional communication is still mainly administrative and billing oriented. As structures and processes in the biomedical domain are subject to *rapid change*, e.g., caused by the introduction of new diagnostic or therapeutic procedures [LK04], the developed concepts have to be robust and adoptable at the same time. As the different domains and groups are focusing on their specific research goals, based on latest and adapted domain knowledge, it is very difficult to effectively standardize or adopt commonly needed data models and structures [KB09].

Security and *privacy* aspects are highly relevant and related to complex ethical, legal, and socio-political questions. Access rights are dynamic and, especially in collaborative structures, may be complex: In health care, access to patient data is restricted to the context of treatment whereas in research, the informed consent is a key principle. The latter requires the documentation of the willingness of a subject to participate in a study, after having been informed about all relevant aspects of the research project [GCP]. Integration systems have to provide solutions which allow to balance between the freedom of research, the rights of patients and the usability of data [AAT08]. On the one hand, researchers in the biomedical domain are willing to share their data but, on the other hand, they are not willing to give up *full control* [WMM$^+$09]. It is therefore often necessary to implement distributed systems, which access data sources in conformance with local authentication models.

1.2 Contributions

Within the context of these domain-specific challenges and requirements, previous integration solutions are not flexible enough (e.g., [Wyn08, WWS$^+$10] or require a huge amount of time and effort (e.g., [ZRGD10]). The aim of this work is to investigate to which extent modern methods from information management can be utilized to efficiently provide and integrate data, information and knowledge for translational medical research. Particular attention is paid to aspects of heterogeneity, volatility and autonomy. The groundwork of this thesis is laid by an integration methodology which allows to utilize the expressiveness of ontologies in an incremental manner within a distributed environment. Secondly, an implementation concept is presented and associated technical challenges are identified. Finally, solutions for these challenges are proposed and a prototypical implementation is presented which provides a comprehensive graphical user interface. In detail, the contributions of this thesis are as follows:

Incremental ontology-based integration: This novel integration methodology is oriented towards the requirements and challenges of the application domain. It utilizes the expressiveness of ontologies to bridge semantic heterogeneity that originates from the distributed and autonomous manner in which data is collected throughout the domain. As the deployment and maintenance of ontology-based solutions often requires a large amount of time and effort, the concept proposes an incremental approach. The basic idea is that an integrated access to co-existing datasets is already useful for researchers. As requirements and datasets in translational research are subject to constant change, further semantic integration efforts are carried out in an incremental and demand-oriented manner. A concept is presented which implements this methodology within a distributed environment based on Semantic Web technologies, i.e., the *Resource Description Framework* (RDF) data model and the SPARQL query language. The resulting system is flexible and built upon loosely coupled components. It avoids unnecessary work and is able to quickly reflect changes.

Access to biomedical data sources: The implementation concept is based on the graph-structured RDF data model. We therefore propose different concepts for transforming biomedical datasets into RDF. Because relational database systems are the dominant storage solution within the domain, the thesis focusses on a novel approach for this data model. In conformance with the overall goals of the methodology, it is centered around flexibility and usability and is solely based within the RDF world. To this end, the approach provides means to apply meaningful transformations to the data which still preserve the original context of the data items. Moreover, a generic approach for integrating messages which conform to the HL7 standard for information exchange between information systems in patient care is presented. It makes use of the fact that the syntax of HL7 is standardized and utilizes a third-party machine-readable representation of its specification to transform messages into an RDF representation. None of the developed techniques requires any upfront knowledge of the schema or semantics of the underlying systems and initial mappings can be generated automatically. RDF databases, such as publicly available knowledge bases and vocabularies, can be directly integrated into the system.

Distributed query processing for RDF databases: Querying a set of distributed RDF databases is one of the key functionalities of our approach. It is not only the primary interface available to applications or end users, but also forms the backbone of the developed concepts for further semantic integration. Therefore, several compile-time and and run-time optimization techniques for the distributed execution of SPARQL queries are proposed. The concept aims at integrating a very diverse set of data sources without the need for upfront schema-level knowledge. To this end, query processing is based on a purely syntactical synopsis which is generated automatically. One of the basic ideas of our approach is to implement loose coupling by spreading the data over several distributed repositories. This separation is also utilized to implement a fine grained permission model, while the ability of the RDF data model to cross-reference other datasets is utilized to "glue" the repositories together. As this inherent distribution strongly increases the complexity of join processing, the presented optimization techniques are specifically targeted on this problem.

Semantic integration and data transformation: Following the incremental and demand-oriented process advocated by the methodology, several approaches for annotating, mapping and transforming data items are presented. This includes an approach for manual data editing, which is feasible within a challenging environment consisting of a large graph structure with millions of nodes. Based on the primary data and user-provided annotations, a comprehensive mechanism for semantic integration is proposed. The basic idea is to provide a scripting language, which can access the underlying data space in an integrated manner via a SPARQL interface. The individual scripts are executed by a workflow engine, which tries to maximize parallelism and to spread the load equally amongst the available computing nodes. The scripting environment provides several operators to post-process the results of these queries and to persist newly generated and transformed data. The provided operators include domain-independent functions (e.g., semantic reasoning) and domain-specific functions (e.g., data anonymization). As privacy is a central requirement within the application domain, we focus on an efficient implementation of a data de-identification operator.

Prototypical implementation: Finally, the feasibility of the approach is shown by presenting a prototype which integrates the previously described components into a comprehensive data integration solution for translational medical research. This includes the description of an overall system architecture which is build upon a data architecture consisting of primitives that are utilized to compose the system and to implement a fine-grained permission model. The prototype includes a graphical user interface for data integrators. This workbench allows to manage all aspects of the solution, to perform annotating- and mapping-tasks, to define data transformation and integration scripts as well as to monitor the overall state of the system.

1.3 Outline

The remainder of this thesis is organized as follows:

- Chapter 2 gives an overview over information integration efforts within the domain of translational medical research. It describes major use-cases as well as challenges and discusses related work. [2]

- Chapter 3 presents the methodology of incremental ontology-based integration, which is solely oriented towards the application domain. It further presents an implementation concept, which forms the groundwork for the approaches and techniques presented in this thesis. [1]

- Chapter 4 describes different approaches for exporting common biomedical data sources into the graph-structured data model underlying the implementation concept. [3]

- Chapter 5 covers distributed query processing and presents a novel index structure as well as various compile-time and run-time optimization techniques. [4]

- Chapter 6 presents a comprehensive approach to semantic integration and data transformation, which builds upon a dedicated workflow engine. The chapter focusses on the de-identification of biomedical datasets. [5]

[2]Parts of the work presented in this chapter appeared in [PWL+11b] and [PWL+11a]
[3]Parts of the work presented in this chapter appeared in [PWL+11a] and [PKKK12]
[4]Parts of the work presented in this chapter appeared in [PWL+11a] and [PKK12]
[5]Parts of the work presented in this chapter appeared in [KPE+12b] and [KPE+12a]

- Chapter 7 describes a prototype implementation, which unifies the developed techniques and provides an integrated graphical user interface oriented towards data integrators.
- Chapter 8 concludes the thesis.

CHAPTER 2

The Challenge: Integrating Data for Translational Medical Research

This section provides an overview over information integration efforts for translational medical research. It describes common use cases and covers domain-specific challenges and requirements. Finally, we discuss related work which includes important infrastructure projects, approaches oriented towards clinical data and solutions from the area of bioinformatics.

2.1 Use Cases

There are several ways in which a comprehensive domain-specific information integration solution can support translational medical research processes. Typical use cases include the annotation, linkage and mapping of phenotypes and genotypes from research or clinical systems, the provision of data for statistical analyses and the integration of knowledge bases. Clinical trials can be supported by assessing their feasibility through estimating the potential number of patients with a predefined profile, which attend an institution during a certain time period. In the following, these important use cases will be explained in more detail.

2.1.1 Data Retrieval and Integration of Knowledge Bases

Due to the complexity of relevant data and processes, knowledge-based methods are of high relevance for translational medical research. Therefore, the provision and integration of knowledge from different domains and scientific areas is an important requirement for several applications. Major data sources of interest comprise knowledge bases with biomolecular knowledge, domain models and ontologies, as well as (cross-domain) metadata and terminologies. Many of these data are graph-structured, including, e.g., metabolic pathways or interaction networks (e.g., gene-gene-, gene-protein- or protein-protein-interactions) from molecular biology. Figure 2.1 presents two examples for such graph-structured datasets. It shows an excerpt of RxNorm [LMM$^+$05], which is a standardized nomenclature for pharmacology, as well as a subgraph of a metabolic pathway from KEGG (Kyoto Encyclopedia of Genes and Genomes) [KG00], which is a knowledge base for the systematic analysis of gene functions.

A domain-specific integration solution must be able to integrate a very diverse set of knowledge bases. Furthermore, it needs to provide means to define mappings and link the knowledge to data from other contexts, such as research databases, biobanks or clinical systems. In this area, the important role of modern data management concepts has been widely understood. For example, in [Sar10] the author states that „[the] need to identify relevant information from multiple heterogeneous data sources is inherent in translational medicine [...]. [...] information retrieval systems could be built on existing and emerging approaches from within the biomedical informatics community, including [...] *Semantic Web* technologies".

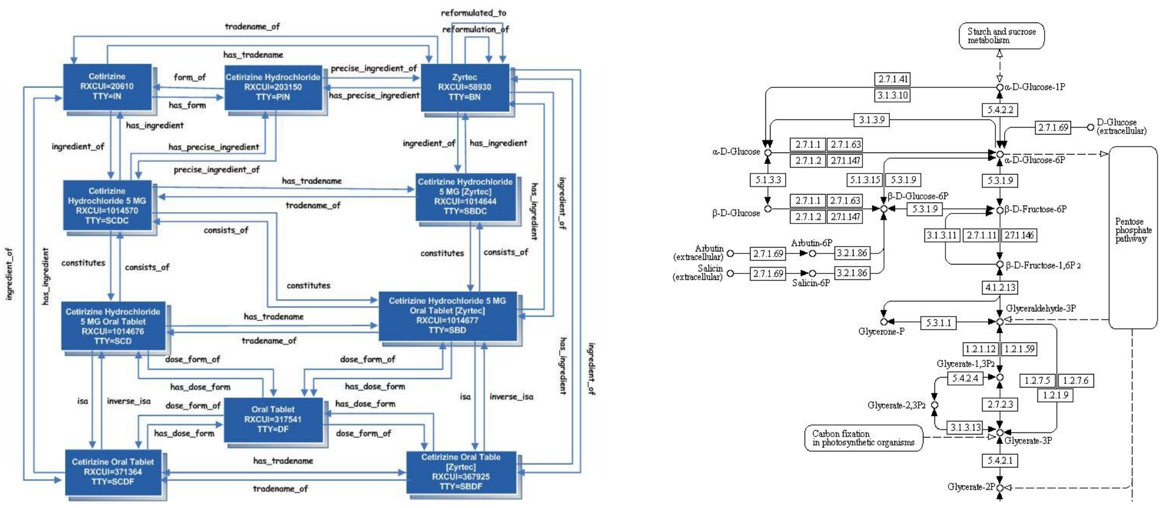

Cetirizine RXCUI=20610 TTY=IN
Cetirizine Hydrochloride RXCUI=203150 TTY=PIN
Zyrtec RXCUI=58930 TTY=BN
Cetirizine Hydrochloride 5 MG RXCUI=1014570 TTY=SCDC
Cetirizine Hydrochloride 5 MG [Zyrtec] RXCUI=1014644 TTY=SBDC
Cetirizine Hydrochloride 5 MG Oral Tablet RXCUI=1014676 TTY=SCD
Cetirizine Hydrochloride 5 MG Oral Tablet [Zyrtec] RXCUI=1014677 TTY=SBD
Oral Tablet RXCUI=317541 TTY=DF
Cetirizine Oral Tablet RXCUI=371364 TTY=SCDF
Cetirizine Oral Table [Zyrtec] RXCUI=367925 TTY=SBDF
tradename_of
has_tradename
form_of
has_form
precise_ingredient_of
has_precise_ingredient
reformulated_to
reformulation_of
ingredient_of
has_ingredient
consists_of
constitutes
dose_form_of
has_dose_form
isa
inverse_isa
Starch and sucrose metabolism
α-D-Glucose-1P
α-D-Glucose
α-D-Glucose-6P
β-D-Glucose
β-D-Glucose-6P
β-D-Fructose-6P
β-D-Fructose-1,6P2
D-Glucose (extracellular)
Arbutin (extracellular)
Arbutin-6P
Salicin (extracellular)
Salicin-6P
Pentose phosphate pathway
Glyceraldehyde-3P
Glycerone-P
Glycerate-1,3P2
Glycerate-2,3P2
Glycerate-3P
Glycerate-2P
Carbon fixation in photosynthetic organisms

Figure 2.1: Graph structured biomedical data [LMM+05, KG00]

These individual building blocks ultimately aim at the provision of integrated information for conducting statistical analyses. Here, an integration solution also needs to support research-specific, highly structured data collections. These research databases are often built by reusing (i.e., transferring and annotating) data which has been collected in the context of patient care (secondary use). After a data collection phase, hypotheses are generated and tested based on complex relationships between demographic, phenotypic and biomolecular parameters. This requires the provision of a querying language and data transformation mechanisms, which allow to represent complex relationships and to evaluate complex expressions.

2.1.2 IT-Support for Patient Recruitment

Patient recruitment is a well-known bottleneck when conducting clinical trials that can lead to significant delays [CSF+07]. There are different ways to support clinical trails in this context. Firstly, the feasibility of conducting a trial at a specific location can be evaluated by estimating whether the required number of suitable trial subjects will be available within a predefined timeframe. Secondly, patient recruitment can be supported actively by providing means to query for potential study participants based on structured information or to implement an alert mechanism as, e.g., demonstrated in [EJC+05, Mil06, WBHF03].

Patient recruitment is based upon a catalog of inclusion and exclusion criteria, which have to hold for any potential participant. Suitable candidates have to fulfil all inclusion and none of the exclusion criteria. These criteria are normally provided in an unstructured manner (i.e., text) and are part of a trial's study protocol, which defines the process of conducting the study. Clinical trials are classified according to their phase. The purpose of a phase-1 trial is to investigate the tolerability, absorption and mode of action of a substance with a small number of study participants. A phase-2 trial investigates the efficacy of a substance for the treatment of specific diseases as well as the optimal dose and side effects. The goal of a phase-3 trial is to statistically prove the efficacy and tolerability of a substance with a larger cohort of thousands of voluntary participants including a control group (placebo). Finally, a phase-4 trial investigates the broader application of a substance and tries to confirm risk-benefit analyses and find rare side effects.

The feasibility of estimating patient accrual rates and performing cohort selection based on electronic information has been confirmed in several publications, e.g., [DLMT+10, OK07]. The authors of [DMM09] analyzed relevant queries issued by researchers within a timeframe of three years and determined the most frequent categories of information. The most important types of data were demographics, diagnoses, procedures, laboratory results and medication. This shows that clinical trials can be successfully supported by utilizing structured electronic information from an ADT system, Clinical Trial Management Systems, Electronic Medical Records and further data from Laboratory and Pathology Information Systems. As eligibility criteria are often very complex [RTCS10], a patient cohort selection tool can benefit from providing a highly expressive querying mechanism on semantically rich data. In addition, the system needs to provide means for complex data transformations, access to metadata and the ability to model temporal constraints [DMM09]. For the automated evaluation of eligibility criteria, models for computable criteria as well as transformation techniques for free-text criteria are being developed [TPC+11].

2.2 Challenges and Requirements

There are several domain-specific challenges and requirements, which need to be considered when implementing the described use cases. Firstly, there are well-known challenges for integration solutions themselves, which are particularly prevalent in the biomedical domain. Secondly, the continuous evolution within the domain demands for robust and flexible approaches. Finally, privacy and security aspects are of central importance.

2.2.1 Distribution, Autonomy, Heterogeneity

Distribution and autonomy as well as the resulting heterogeneity are well-known challenges for information integration systems and solutions are often characterized along these orthogonal axes [SL90]. As translational medical research requires to integrate data across domain and institution boundaries, these challenges are particularly prevalent. The different domains, some of which are themselves fragmented, collect and manage data in different ways, e.g., due to different operational procedures, cultures or legal and regulatory requirements. Therefore, relevant data is highly heterogeneous in terms of their underlying data models, structure, vocabularies and semantics.

Throughout the domain, controlled vocabularies are only in limited use. While in the area of bioinformatics standard terminologies are more wide-spread, they are mostly used for billing purposes in patient care. Moreover, in the clinical context, essential findings are often represented in free text, which, although following a certain organization, only implicitly represents structure and vocabularies. If structured forms are being used, they are often not harmonized between different departments. In research, structured forms and standardized terminologies are more prevalent, but still the integration of data collected in different studies tends to need substantial efforts and harmonization [WLP+09, FBR+10]. A reliable and secure management of patient identities across different institutions is often missing. Even to attribute all existing data to a specific patient within one institution is often a problem. Modern analytical biomolecular methods like genomic, transcriptomic, proteomic, or metabolomic analyses produce large volumes of highly complex data. Scientific progress has reached a point at which background knowledge is essential to effectively understand and interpret results [PES09]. The relational model is well suited for representing structured data from health care and clinical research, but other data models are also relevant, e.g., for managing biomolecular data or ontologies.

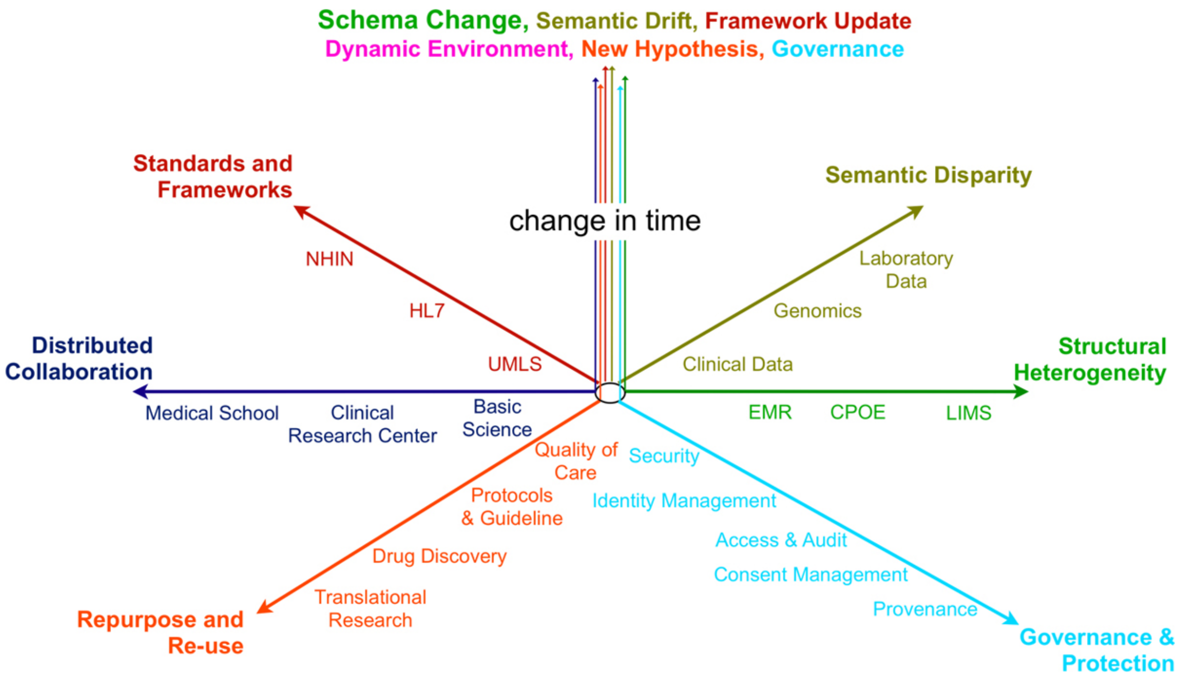


Figure 2.2: Continuous evolution within the biomedical domain [MZV+09]

These challenges show that an information integration solution for translational medical research must be based upon a highly flexible data model which is able to represent heterogeneous data with different degrees of structure. The integration of knowledge has to be enabled by providing means to enrich data with semantics and integrate them with knowledge bases [Sar10]. The integration of diverse types of data requires an expressive data model which is able to represent complex semantic relationships amongst primary data, knowledge bases, ontologies, metadata and terminologies. To this end, it is also necessary to implement powerful techniques for semantic integration which allow to bridge semantic gaps. On an architectural level, a solution has to be able to integrate data sources, which are distributed amongst different technical and organizational structures. Researchers in the biomedical domain are often willing to share their data but, on the other hand, they are not willing to give up full control over their datasets [WMM+09]. Therefore, the access autonomy of local data sources must not be compromised, which requires distributed systems that access the local data sources only on demand and in conformance with local authorization models.

2.2.2 Continuous Evolution

System requirements in the biomedical domain are subject to rapid change. Already in health care environments, medical structures and processes are continuously evolving due to new diagnostic or therapeutic procedures [LK04]. Complexity is increased by the descriptive nature of life sciences, especially biology, which lack an underlying mathematical model [KB09, Laz02]. This situation leads to unique challenges for software development [KB09]. Even fundamental definitions of the discipline, e.g. the definition of „gene", can change, making long-term assumptions for data structures unreliable. Therefore the question is often what to implement, not how to implement it and research in life sciences has even been called „by nature, borderline chaotic" [KB09].

In the medical domain, the fact that experts are typically inept to describe the processes and requirements of their own work, leads to difficulties in software engineering [WB05, ADB04, FW97]. This is aggravated by the well-known circumstance of interdependence between social and technical aspects, in which the introduction of IT solutions changes workflows and therefore requirements [WB05,LK04]. Researchers tend to exclusively focus on their specific biomedical research projects, with very little understanding of technical aspects and IT concepts. Activities with immediate visibility are prioritized over sustainability, especially in a domain where time to publication is of utmost importance [KB09]. Therefore „even a software considered a failure by developers can be considered a success by scientists" [KB09] and the other way around.

Figure 2.2 shows an overview over six different axes, which influence the requirements for integration solutions in the biomedical domain. The presented aspects are continuously changing over time. For example, standards and frameworks are being updated, the required data is changing due to new hypotheses, information systems are undergoing frequent schema updates and legal as well as regulatory requirements are constantly evolving. Therefore, integration solutions have to be very flexible and robust in order to adopt to new requirements.

2.2.3 Data Privacy and Security

Privacy and security are key non-functional requirements in translational medical research, which lead to challenges at the ethical, legal and socio-political level. For information management, anonymization and pseudonymization based on the informed consent of the patient are central concepts for data protection. This includes the patient's right to withdraw the consent at any time without consequences for treatment, possibly leading to the need for the deletion of data. According to the German Federal Data Protection Act, anonymization means „[...] the alteration of personal data in a way that information concerning personal or material circumstances cannot be attributed to an identified or identifiable natural person or that such attribution would require a disproportionate amount of time, expense and effort" [DPA]. Pseudonymization is defined as „[...] replacing the data subject's name and other identifying features with another identifier in order to render it impossible or extremely difficult to identify the data subject." [DPA].

It is important to realize that it is often not sufficient to only remove directly identifying information, such as *name* or *address*. In [Swe02] the author purchased the voter registration list for Cambridge, MA and linked it with presumably anonymous medical data released by the Group Insurance Commission of Massachusetts. While the voter list contained *name*, *address*, *zip code*, *date of birth* and *gender* of the voters, the medical data covered *visit dates*, *diagnoses*, *procedures*, *zip code*, *date of birth*, and *gender*. By linking the latter three attributes, she was able to re-identify the medical record of the current governor of Massachusetts. In an earlier study the author had shown that the attributes *zip code*, *date of birth* and *gender* uniquely identify about 87% of the U.S. population. This problem becomes even more prevalent when different data from diverse sources is being integrated [HSR+08].

The dynamic and cooperative structure of access rights also poses challenges for new and innovative solutions. Without informed consent, access to patient data is only allowed in the context of treatment, which changes frequently. This complex situation of permissions and roles, as well as the researchers' intellectual property rights have to be considered in any integration solution. The regulatory landscape as well as local, national and international laws and regulations have to be reflected.

For integration systems these challenges require to implement complex authorization concepts, which ensure that all regulatory requirements are fulfilled and that researchers maintain full control over their local datasets. To balance the individual's privacy and the researchers' needs for high-quality data, techniques need to be incorporated which allow to anonymize the integrated datasets with minimal information loss. Pseudonymization concepts are required to ensure a maximum degree of data protection and to implement fine-grained authorization methods, which allow to separate access to different types of data, but preserve the option of putting data back into its original context.

2.3 Related Work

For the retrospective integration of distributed, heterogeneous data collections several architectural solutions have been developed over the past decades. Most of these are also being applied in the biomedical domain [LMMS+07]. This includes data warehousing approaches as well as virtual information integration systems, some of which are built on top of object-oriented middleware solutions. Major research programs have led to an application and a refinement of these concepts. One of the research programs, the Clinical and Translational Science Awards (CTSA) program of the NIH/NCRR, aims at building environments for clinical and translational research [CTS]. Thus, centers with sophisticated biomedical informatics infrastructures have been established throughout the US. Projects with objectives similar to those of CTSA are being funded in the EU, e.g., [MPL07, SIM].

The „cancer Biomedical Informatics Grid“ (caBIG) is an intiative of the US National Cancer Institute (NCI) [Wor07]. Its goal is to provide tools for communication and colaboration in translational cancer research. The project is separated into different domain-specific and cross-domain workspaces. Amongst others, the domain-specific workspaces include clinical trial management systems (CTMS), biobanks and tools for pathology (TBPT) and integrative cancer research (ICR). The cross-domain workspaces include data standards and vocabularies (VCDE), standards for software architectures (ARCH) and strategic planning (SP). Some of the components developed in the context of caBIG are frequently re-used in other projects, such as caDSR (a metadata repository for common data elements) or caGrid (a grid-middleware designed for data exchange and distributed query processing).

At Harvard Medical School, cooperating with Partners HealthCare, physical repositories exist for intra-institutional integration of data for patient care, quality assurance and research. The „Research Patient Data Repository" (RPDR) implements a data warehouse and stores data of 4.6 million patients consisting of 1.2 billion diagnoses, procedures and laboratory results. To ensure the patients' privacy, these data are de-identified and access is controlled by an institutional review board (IRB, an independent committee which is responsible for the protection of human subjects in research [GCP]). After a preliminary enquiry which allows access to aggregated data only, the review board can also grant access to the complete result set of a query [Wur11]. Additional tools for data processing and analysis are offered based on „Informatics for Integrating Biology and the Bedside" (i2b2) [MWM+10]. i2b2 allows researchers to import de-identified patient data into their own repository where they are allowed to alter it in any way they wish without interfering with other analysis and cleansing processes. Heterogeneity and change is addressed by building upon a flexible star schema that reflects properties of the generic EAV (entity-attribute-value) schema approach [NMC+99]. Concepts can be represented by referencing vocabularies, terminologies or ontologies. In [DMM09] it has been shown that i2b2 still suffers from some limitations when used for patient cohort selection, such as querying for temporal conditions. An integrated view over three i2b2 instances at the different the Harvard clinics is provided by a federated query tool, called SHRINE [WMM+09].

Vanderbilt University in Nashville TN is also pursuing a typical replication approach for intra-institutional integration. For this purpose, data originating from the electronic medical record (EMR) system of the clinic are replicated and linked with a biobank (BioVU). For privacy reasons the contained clinical data is de-identified. De-identification is achieved by shifting all dates within patient records, removing potentially identifying attributes and applying a one-way hash to patient identifiers. The date shift is different between records but constant within one patient's record. The removal of the potentially identifying attributes is in compliance with the HIPAA privacy rules. HIPAA is the US Health Insurance Portability and Accountability Act from 1996 which contains a section listing 17 attributes that should be removed from a patient record for de-identification. The removal is automatically performed using a NLP (Natural Language Processing) tool. The one way hash is used for linking the research data with biosamples from BioVU and prospectively enables the addition of follow-up data from the EMR-system without the danger of re-identification [RPB+08].

An example for virtual information integration built on-top of an object-oriented middleware can be found at the University of Utah. The Federated Utah Research and Translational Health e-Repository (FURTHeR) implements inter-institutional integration by spanning an object-oriented logical data model over disparate data sources. Federated query processing is implemented based on a global in-memory database. It integrates Utah's largest patient data repositories (University of Utah, Intermountain Healthcare, Salt Lake City Veterans Administration Medical Center), public health resources from the Utah Department of Health as well as genealogic and demographic data from the Utah Population Database [BML+09]. Local permission models are not compromised as FURTHeR is integrated with local electronic IRB systems [HHBN10].

There are several other projects, which are often more oriented towards bioinformatics use-cases. Some of them utilize components from caBIG or i2b2. SlimPrim implements a central repository for the integration of de-identified clinical and biomedical data [VBS+09, VBV+09]. Anduril is a bioinformatics tool which implements a central repository that is able to execute data analysis workflows, which integrate external components and databases [OLHP+10]. TranSMART is a translational data warehouse which is implemented on top of i2b2 and runs in a cloud computing environment [SKKP10, PVS10]. Triton is a data integration solution that has been evaluated for chronic lymphocytic leukemia research and builds upon the caBIG toolchain [PBS+10]. Galaxy is a web-based workbench for analyzing and integration genomic data and provides a workflow engine [GNTT10]. iDASH is a tool for sharing data in a privacy preserving manner [OMBB+12]. Atlas is a bioinformatics data warehouse which integrates different biomolecular datasets [SHX+05].

Ontology-based approaches for information integration in the context of translational medical research are a promising new development [Qua07, RCB+07]. Examples can be found at the Health Science Center at the University of Texas in Houston [MZV+09] and at the University of California, San Francisco [Wyn08]. Other projects integrate i2b2 and caBIG with ontology-based concepts [MPB+09, SCT+10] or are based upon the RDF data model [NBHH08,SCY+07]. As the solution presented in this thesis is also ontology-based, this related work is covered in more detail in Section 3.1.

In addition to distribution, integration architectures have to deal with semantic heterogeneity. This is implemented based on canonical or standardized data formats and common data models. Already Payne et al. have characterized semantic interoperability as the fundamental challenge for informatics in the context of clinical and translational sciences [PES09]. Although many competing standards have been developed over time, standards that enable the sharing of information between basic sciences, clinical science and clinical practice have yet to be developed [EP09]. In this context, current projects are mainly focusing on extensive data exchange between information systems in the context of treatment and clinical research. Various standards, which partially build upon each other, have been developed for clinical data, such as the HL7 V3 Reference Information Model (RIM), the HL7 Clinical Document Architecture (CDA), the Continuity of Care Record (CCR) [CCR] and the Continuity of Care Document (CCD). Further standards exist for information exchange in the context of clinical trials, such as the Operational Data Model (ODM) [CDI]. Under the umbrella of caBIG a standardization of data elements (Common Data Elements, CDEs) for capturing and exchanging information in translational research is being developed [Wor07]. It follows the rules of the ISO / IEC 11179 standard for metadata repositories and aims at covering the complete spectrum from patient treatment to clinical and basic sciences. The „Biomedical Research Integrated Domain Group" (BRIDG) is an ambitious effort to harmonize various data standards from healthcare as well as clinical research and has been set up as a joined collaboration [FEHM08].

CHAPTER 3

The Solution Concept: Incremental Ontology-based Integration

Ontology-based approaches implement data integration with highly expressive data models that provide well-defined explicit semantics. A relatively new development in the context of translational research, they carry a huge potential for the integration of heterogeneous biomedical data. This chapter presents an abstract integration methodology, which utilizes the flexibility of this approach to incrementally integrate heterogeneous datasets within a distributed environment. It further presents an implementation concept which aims at leveraging its inherent flexibility to easily adjust to changes in data, metadata, use-cases and surrounding conditions.

3.1 Related Work

The proposed integration methodology is built upon key ideas from two different research areas. Firstly, ontology-based integration provides means to integrate highly heterogeneous datasets. Secondly, the dataspaces concept envisions integration solutions which are centered around the idea of data co-existence and implement an incremental approach to semantic integration. The ultimate goal is the ability to integrate many, highly heterogeneous data sources within a constantly evolving environment. This section presents related work from both of these areas.

3.1.1 Ontology-based Integration

In computer science an ontology is normally defined as an „explicit formal specification of a conceptualization" [Gru93]. This means that it is a formally defined system of concepts and their relationships. The statements within an ontology can contain information on meta- or type-level (terminological component, or TBox) as well as instance-level information (assertion component, or ABox). As these different information are tightly coupled, an ontology can often – depending on its specific implementation – be represented as a network of objects, i.e., a directed labeled graph. An ontology is also allowed to contain instance-level information that is not related to the contained meta-level data items. In information integration, ontologies are traditionally utilized as a supportive resource for specific aspects of semantic integration (e.g., schema matching). In contrast, ontology-based integration is a more holistic concept in which ontologies are utilized to overcome many different types of semantic heterogeneity on data and schema level, i.e., structural conflicts, data model conflicts, naming conflicts and to some extend even syntactical conflicts. To this end, primary data is represented in a common ontological system, e.g., a large directed graph, which means that the separation of data and metadata (type- and instance-level) disappears. Ontology-based integration is mostly understood as a top-down process: Firstly, a context-specific ontology is designed which takes the role of a global schema. Secondly, the heterogeneous datasets (i.e., its data items and concepts) are related (i.e., mapped) to this global ontology. Further semantic relationships between the data from the different systems can then be discovered by applying inference algorithms. Alternatively, the local datasets can be mapped to their own local ontologies, which are then harmonized in an additional ontology alignment step. A hybrid approach would combine the use of local ontologies and a global ontology. A comprehensive overview of the different conceptual design alternatives is given in [WVV+01].

At the Health Science Center of the University of Texas in Houston an infrastructure has been developed in which different integration tasks as well as further requirements are solved by utilizing one single ontology [MZV+09]. The solution includes concepts for authentication, rights management, concept-based navigation and a workflow for designing Case Report Forms (CRFs). To this end, the explicit semantics of ontologies is utilized to implement the described functionality based on a single data representation. The developed ontology is structured into different layers which have been built on top of each other. Each model covers a specific set of aspects of the application domain. For example, the „Integrated Vocabulary Model" harmonizes different knowledge representation systems (e.g., UMLS) by transforming them into a SKOS representation. SKOS (RDF: Simple Knowledge Organization System) is a schema for the definition of controlled vocabularies. The resulting global ontology is then processed by a reasoner which is able to infer correspondences between concepts from the different data sources. Although the separation into individual layers simplifies the management of ontologies, ontology-based integration remains work-intensive.

Another ontology-based approach has been developed at the University of California, San Francisco. Here, the basic idea is that an ontology-based system is able to react to constant change, to integrate highly heterogeneous data and to provide different views on the integrated data to different groups of users [Wyn08]. Again, its expressiveness and its ability to integrate highly heterogeneous datasets in a flexible manner motivates the use of an ontology-centric approach. The system aims at integrating more than 50 complex data sources from different disciplines. Due to the high effort, a comprehensive mapping onto a global ontology is avoided. Instead, the original data is processed by an expert system, which performs semantic integration on-demand. This allows to provide different relationships between the data elements to different users at runtime. This methodology has been implemented in [WWS+10] and is related to the concept that is presented within this work (see Section 3.2).

In [MPB+09] a system has been implemented which integrates solutions provided by the caBIG project by mapping the underlying UML model to an ontology formulated in the Web Ontology Language (OWL) [OWLa]. OWL is a vocabulary for the graph-structured Resource Description Framework (RDF) data model [MM04], which forms the basis of many ontology-based applications (see Section 3.2.2 for more details). In [SCT+10] data federation is implemented over human study databases based upon caBIG tools, i2b2 and a research-related OWL ontology. TIM is an ontology-based application for the management and specification of metadata items in clinical research [MKM09]. Highly specialized approaches include [ABB+07,CVF+06,BEF10]. Some solutions are not explicitly ontology-based but utilize the RDF data model. BioMANTA implements a central RDF triple store for biomedical data based upon cloud computing technologies [NBHH08]. LinkHub integrates different RDF graphs and is oriented towards genomic research [SCY+07].

3.1.2 Dataspaces

Dataspaces [FHM05] have been proposed for the integration of many, highly heterogeneous data sources without the need for large upfront efforts such as schema integration. Instead, the integrated access to coexisting data is the central concept. Additional semantic integration is carried out in a demand-oriented and incremental manner („pay-as-you-go" integration). This stands in contrast to established integration approaches which require an initial schema integration. Dataspaces have been designed for the efficient integration of data sources with different degrees of structure. An additional advantage is that the incremental methodology also increases the ability to react to changing requirements [WLP+09]. This is shown in Figure 3.1, which presents a conceptual comparison of the incremental and the schema-first approach. The schema-first approach is characterized by an initial ramp-up phase in which the data is being semantically integrated. In contrast, the incremental approach already provides some initial functionality, which is then incrementally extended to better meet the requirements. Within a volatile and rapidly evolving domain, it is necessary to repeatedly adjust the schema-first approach to prevent a decrease in functionality. As the incremental approach has been designed to evolve constantly, it has the potential to better adjust to this conditions.

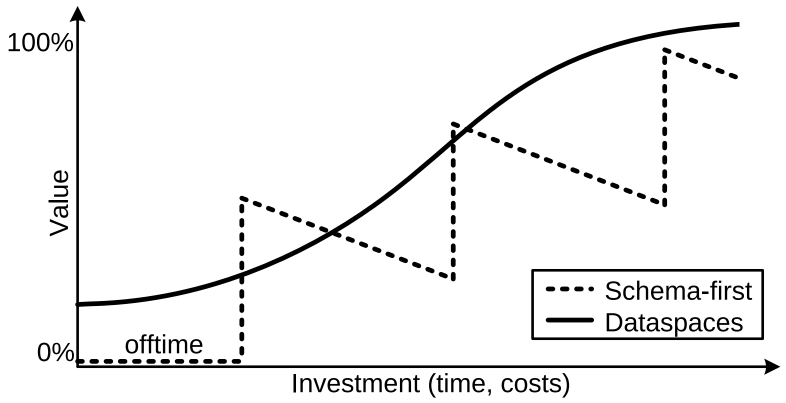


Figure 3.1: Incremental vs. schema-first in a volatile domain [WLP+09]

The idea of dataspaces is only an abstract concept. Therefore, it has been implemented in many different ways and for different application scenarios. What stands out is that many implementations are based upon a graph-structured data model. This is because graphs are highly flexible and well suited for representing data with different degrees of structure. For example, the system described in [DS06] implements a generic transformation of all data sources into a large, directed graph structure. Incremental semantic integration is implemented by defining type- and instance-level rules which describe dependencies among the nodes and edges in the graph. Other examples for approaches which implement the dataspace paradigm based upon graph-structures can be found in [Biz09] and [WLP+09].

Further research has been carried out along the principle design axes of dataspace systems, which have been presented in [HFM06]. This includes dedicated index structures [DH07, HMRR08, ND08], metadata models [JB04, CCSS07, Mus08], probabilistic and best-effort services [Don07, DDH08], utilizing user feedback [JFH08, BPE+10, BPF+11] and systems oriented towards web data [TWH09, MJC+07, CRS+07, JDG07, TWH09]. For translational medical research, a data extraction system has been proposed which focusses on preserving local access autonomies [WLP+09]. Incremental semantic integration is implemented to better handle the continuously evolving application domain and access autonomies are preserved by implementing a distributed mediator/wrapper architecture. The wrappers are designed to leverage local authentication and authorization mechanisms via suitable system interfaces and directly access the underlying databases [Wur11]. The system is oriented towards exchanging patient-centric information to support the re-use of clinical data for research.

3.2 Incremental Ontology-based Integration

This section first presents a generic concept which unifies ontology-based integration and the dataspace paradigm. The goal is to define a methodology which is able to integrate diverse datasets by utilizing ontology-based concepts in an incremental manner. Secondly, an implementation concept is proposed which is solely oriented towards the specific requirements and challenges of the application domain. This concept is based upon the graph-structured Resource Description Framework (RDF) data model. Therefore, RDF and its most relevant properties will be described in detail.

3.2.1 Basic Ideas

Implementing ontology-based concepts for translational medical research is a challenging task. This is mainly due to the need to provide comprehensive semantic integration within a constantly evolving domain. The tight coupling of the layers involved (data sources, local ontologies, global ontology) leads to a fragile system, which is not robust against constantly changing data and metadata and thus requires frequent adjustments by domain experts. Moreover, ontology-based solutions generally require a schematic description of each data source, which makes integrating data with an unknown, unpredictable or unreliable structure impossible. On the other hand, the fact that the underlying data models are often schema-free makes them especially suitable for such data. In order to overcome these limitations, ontology-based concepts can be combined with the dataspace paradigm.

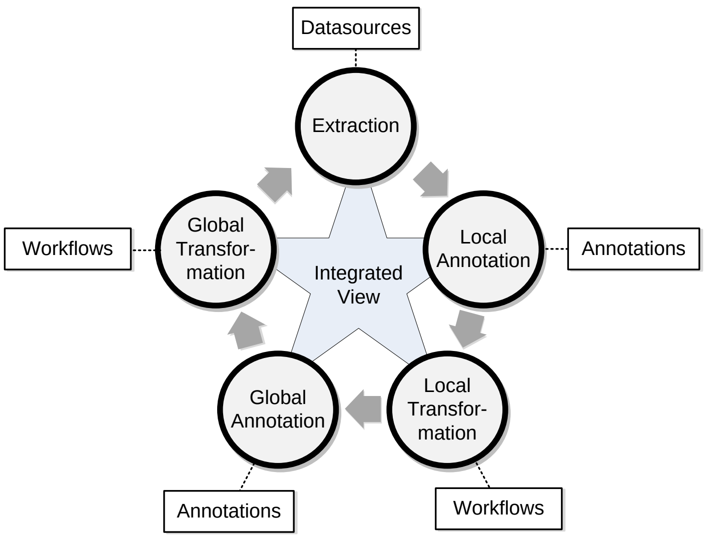


Figure 3.2: Incremental ontology-based integration workflow

Incremental ontology-based integration defines a concept in which ontology-based integration is carried out iteratively without a static top-down or bottom-up methodology. Data sources are transformed into a (probably less meaningful) graph structure in a (semi-) automatic manner. Initially, a meta-level description does only exist as far as it can be derived automatically. The resulting global ontology covers type- and instance-level information, but is characterized by heterogeneity as it lacks semantic integration. On the other hand, it already contains all of the required primary data. In a second step, the resulting semantic graph structure is integrated in an iterative and demand-oriented manner by manually defining local and global annotations as well as data transformation workflows. These workflows can either integrate local data items (i.e., define relationships between data items within a single dataset) or global information (i.e., generate relationships between data items from different sources). Because of the generic data model, differently structured datasets can be mapped into the global dataspace and integrated incrementally. As a result, the individual data sources and metadata are only loosely coupled; changes within one data source might affect small parts of the dataspace, but the rest remains unchanged within the global ontology. As can be seen in Figure 3.2, these individual steps can be interwoven and repeated in an arbitrary manner. If relevant data is too heterogeneous for a certain use case, it is possible to evolve the ontology accordingly without affecting any of the previous applications. The system remains robust to changes.

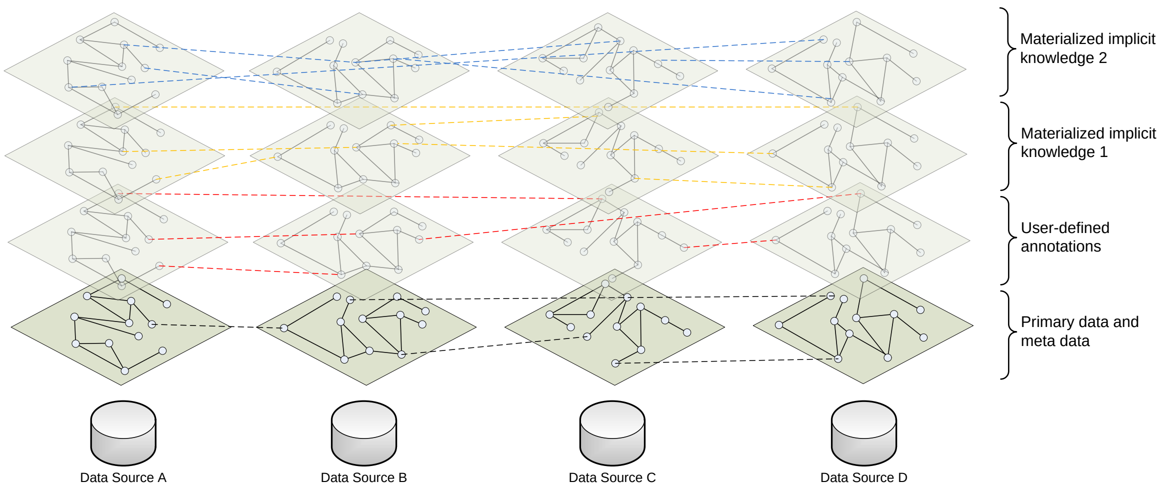

Materialized implicit knowledge 2
Materialized implicit knowledge 1
User-defined annotations
Primary data and meta data
Data Source A
Data Source B
Data Source C
Data Source D

Figure 3.3: Layered representation of explicit and implicit knowledge

Loose coupling within the system is supported by its distribution. This also allows to transparently integrate domain-specific integration primitives, such as a Master-Patient-Index (MPI), terminologies or thesauri, as separate „data sources" into the dataspace. Data transformation workflows are not only able to represent classical relationships, such as the transitive closure of class inheritances, but can also implement domain-specific tasks. For example, a workflow can materialize the equivalence relationship for local patient entities, which are mapped onto a common global identifier by the master patient index. Transformation workflows are used to define statements about a set of objects, whereas annotations are utilized to add individual data items to the global ontology. Transformations as well as annotations are always managed separately from the primary data. Therefore, the individual components can (to a large extend) be added and altered independently of each other. Additionally, different workflows, annotations and metadata can be defined for different systems, groups of users or projects. Due to this loose coupling and separation, the resulting dataspace can be seen as a set of layers as shown in Figure 3.3. Here, the primary data exported by the data sources is shown on Layer 1. The additional layers represent user-defined annotations or implicit knowledge which is materialized by executing data transformation workflows. This is similar to intensional associations, a concept which has been proposed for dataspace applications in [SDB10]. All layers can be developed incrementally and independent of each other.

3.2.2 Semantic Web Technologies

The Semantic Web technology stack of the World Wide Web Consortium (W3C) forms the backbone of many ontology-based applications. It basically consists of the *Resource Description Framework* (RDF) data model [MM04] and related technologies such as the query language *SPARQL Protocol and RDF Query Language* (SPARQL) [PS08]. RDF is a graph-structured data model in which information is modelled as a set of triples. Each triple defines an atomic statement of the form (Subject, Predicate, Object) which states that the *Subject* has a property *Predicate* with value *Object*. RDF defines two different types of nodes. Subjects and predicates are always *resources* whereas objects are either resources or *literals*. Resources are identified by globally unique *Uniform Resource Identifiers* (URIs), which are a proper superset of URLs. In order to simplify the representation of structures which consist of several triples (such as lists) RDF allows for anonymous resources whose identifiers are only unique within the local context. Each object is either a resource or a literal. Literals are atomic values with optional type information (e.g., integer or string). RDF uses XML Schema data types, but users are also able to define their own.

Propofol	state	„Liquid“ .
Propofol	formula	„$C_{12}H_{18}O$“ .
Propofol	sideEffect	Apnea .
Apnea	label	„Apnea“ .
Propofol	contraindication	Hypovolemia .
Hypovolemia	label	„Hypovolemia“ .

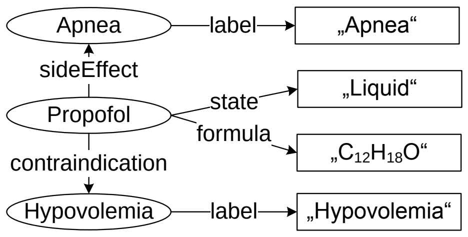


Figure 3.4: Example RDF triples and graph[1]

An RDF graph is a directed labeled graph in which subject and object are labeled nodes and predicates are directed, labeled edges ranging from the subject to the object. Figure 3.4 shows an example set of RDF triples as well as the resulting RDF graph, which encodes information about drugs and their side effects. RDF is also the underlying data model of the concepts presented in this thesis. This section therefore includes a formal definition that follows the work in [PAG09] which has also found its way into the official W3C documents.

```
SELECT ?formula ?sideeffect
WHERE {
  ?compound   contraindication   Hypovolemia .
  ?compound   formula            ?formula .
  ?compound   sideEffect         ?sideeffect .
}
```

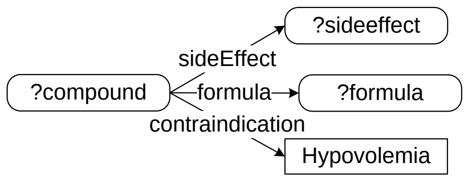


Figure 3.5: Example SPARQL query and Basic Graph Pattern

[1]Troughout this thesis URIs have been abbreviated for better readability

Definition 1 (*RDF Node*)

The set of RDF Nodes T is defined as $I \cup B \cup L$ where

1. *I is an infinite set of URIs,*
2. *B is an infinite set of Blank Node Identifiers,*
3. *L is an infinite set of Literals and*
4. *I, B and L are pairwise disjoint.*

Definition 2 (*RDF Triple*)

An RDF Triple T is a tuple (s,p,o) $\in$ ($I \cup B$) $\times$ I $\times$ ($I \cup B \cup L$).

Definition 3 (*RDF Graph*)

An RDF Graph G is a set of RDF triples.

SPARQL is the standard querying language for RDF data. It is centered around the concept of pattern-matching. The backbone of SPARQL query processing is defined by matching *Basic Graph Patterns* (BGPs) against the queried dataset. A basic graph pattern is defined by a set of *Triple Patterns.* Each triple pattern is an RDF triple in which subject, predicate and object can be substituted by variables. Dependencies between triple patterns are modelled implicitly be utilizing the same variable name. A query which only consists of one such BGP and no further operators is called a *conjunctive* query. Additional SPARQL operators allow to unify the the results from different BGPs (*UNION*), compute an outer join with another BGP (*OPTIONAL*) or define conditions that must hold for the resulting variable bindings (*FILTER*). Finally, results can be projected onto a subset of the contained variables (*SELECT*). As the basic complexity of SPARQL queries is defined by matching the underlying BGPs to the queried RDF graph, we focus on conjunctive SPARQL queries throughout the remainder of this thesis. An example of such a query, which extracts information about the compound Propofol from the RDF dataset shown in Figure 3.4, is shown in Figure Figure 3.5. In the following paragraphs we present a formal definition of conjunctive SPARQL queries.

Definition 4 (*SPARQL Variable*)

The set of Variables V is an infinite set of strings where I, B, L and V are pairwise disjoint.

Definition 5 (*SPARQL Triple Pattern*)

A Triple Pattern TP is a tuple (s,p,o) $\in$ ($I \cup B \cup V$) $\times$ ($I \cup V$) $\times$ ($I \cup B \cup L \cup V$).

Definition 6 (*SPARQL Basic Graph Pattern*)

A Basic Graph Pattern BGP is a set of triple patterns where for each triple pattern $TP_1 \in$ BGP exists another triple pattern $TP_2 \in$ BGP, which shares at least one variable $v \in V$ with TP_1.

Definition 7 (*Conjunctive SPARQL Query*)

A conjunctive SPARQL query Q is a query consisting of exactly one basic graph pattern and no projection.

3.2.3 Important Properties of RDF and SPARQL

The RDF data model offers several properties that render it interesting for a deployment in the biomedical domain. Firstly, it combines flexibility with expressiveness. As data is modeled as a network of objects, RDF is well-suited for the canonical representation of heterogeneous datasets and structures and therefore fosters interoperability. Secondly, RDF provides explicit formal semantics which allow to decompose an RDF dataset into comprehensible atomary statements, even if there is no thorough understanding of the data, e.g., due to missing schema information. Thirdly, RDF enforces the explicit definition of entities, identifiers and relationships. For this reason, under the assumption of a suitable naming convention, resources can be uniquely identified on a global scale and RDF data can be easily combined with information from other datasets. This supports the development of incremental ontology-based approaches to information integration. For example, metadata, annotations or lineage information can be easily added to existing data. Furthermore, new attributes or concepts can be introduced and added to other datasets. At the same time RDF is characterized by its consistency, as data, metadata and semantics can be represented within one model. Although schema information is not necessary for managing RDF data, the *RDF Vocabulary Description Language* (RDF/S) allows to define schema information at any point in time (also incrementally) and to represent it within the RDF model [RDF]. Because of this property, RDF has attracted more and more attention within the bioinformatics domain (e.g., [BNT+08]).

The flexibility of SPARQL is especially useful, if relationships between data items are relevant for answering a query. Additionally, SPARQL supports incremental integration as well as an exploratory interaction paradigm by providing various ways to query semi-structured data or data with an unknown, unpredictable or unreliable structure. To this end, SPARQL allows to query optional relationships or offers means to underspecify attributes or resource-identifiers within queries. This also allows to utilize SPARQL itself (or variuous dialects) for the formulation of simple data transformation workflows for RDF datasets (e.g., [SPI]). Semantic Web technologies carry a lot of potential for solving many of the data management challenges in translational medical research [CFM+09, HCL, SJB+11, Kas11, RCB+07, Qua07]. Until now, Semantic Web technologies are mainly utilized for metadata, annotations and knowledge bases but not for primary data, though. One reason for this is that scalable technologies for the efficient management of very large RDF datasets have only recently been developed. The goal of this work is to utilize the flexibility, consistency and support for distribution of RDF and SPARQL to implement an integration solution which builds upon these recent developments.

3.2.4 Technical Requirements

The access to biomedical data (such as data from RDBMSs) as well as the provision of an integrated view over the distributed sources of information are essential preliminaries for the implementation of the described concept. The utilized methods must be applicable without prior integration, mapping or annotation of the data sources. As a result, lightweight approaches and tools for further incremental semantic integration can be implemented. This includes, e.g., the described annotation and transformation steps, which can be implemented on top of a global ontology-based view on the primary data sources as well as the results of previous integration steps. The global view also serves as an interface for users and applications. The implementation concept builds upon the RDF data model. To incorporate a fine-grained authentication model and preserve local access autonomies it implements a federated approach. From a querying perspective an example of the resulting system design is shown in Figure 3.6.

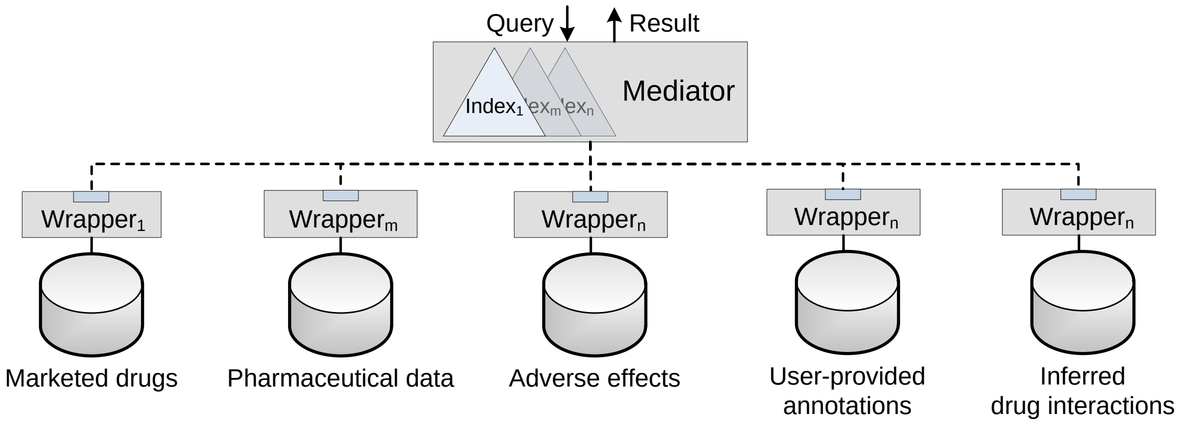


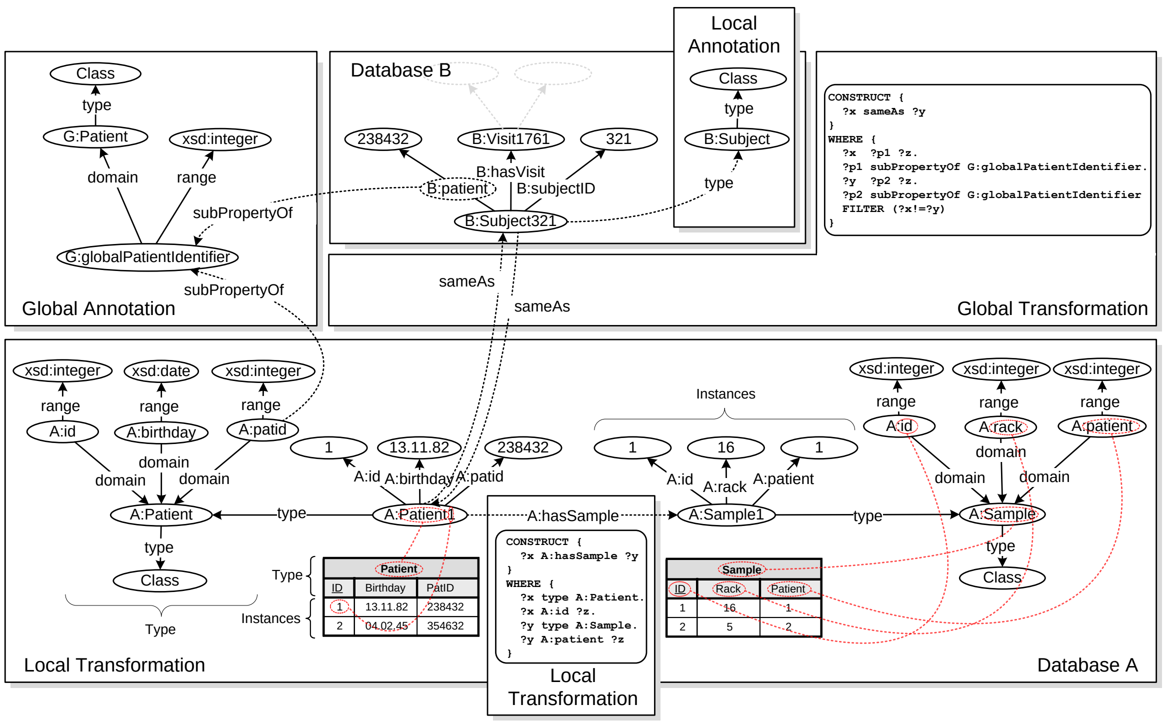
Figure 3.6: Distributed RDF databases

Here, the primary data as well as annotations and materialized implicit knowledge is spread among several distributed databases. The interfaces of those databases are harmonized by a wrapper component which is accessed by a mediator for implementing the required functionalities. A global synopsis allows the mediator to determine relevant databases when executing queries or data transformation workflows. In order to implement this concept, components are required which allow to integrate non-RDF databases into the resulting global graph, provide efficient distributed query processing and execute flexible data transformation workflows. These components need to be

- fully functional within a distributed environment and allow to re-model local authorization models,
- lightweight in terms of deployment and maintenance efforts,
- fully support the described incremental integration process, and
- able to manage large data volumes (several 100 M triples) efficiently.

Example Scenario

An example scenario which can be implemented with the described components is shown in Figure 3.7. Here, type-level information is provided by an RDF vocabulary which is oriented towards RDF Schema. Each triple is a result of the integration step in which its predicate identifier is located. Triples which relate data from different sources are indicated by dashed lines. The access to the relational data source *A* is implemented as a generic database-centric transformation similar to the technique presented in Chapter 4. The relations *Patient* and *Sample* are automatically transformed



```
CONSTRUCT {
  ?x sameAs ?y
}
WHERE {
  ?x  ?p1 ?z.
  ?p1 subPropertyOf G:globalPatientIdentifier.
  ?y  ?p2 ?z.
  ?p2 subPropertyOf G:globalPatientIdentifier
  FILTER (?x!=?y)
}
```

```
CONSTRUCT {
  ?x A:hasSample ?y
}
WHERE {
  ?x type A:Patient.
  ?x A:id ?z.
  ?y type A:Sample.
  ?y A:patient ?z
}
```

Patient		
ID	Birthday	PatID
1	13.11.82	238432
2	04.02.45	354632

Sample		
ID	Rack	Patient
1	16	1
2	5	2

Figure 3.7: A simplified scenario implemented with Semantic Web technologies

into RDF data which covers type- and instance-level. The red dashed lines indicate the relationship between the relational and the RDF representation of these data items. It is assumed that the relationship between the tables can not be automatically transformed to the RDF representation, because the according foreign-key constraint is not defined in the database's metadata catalog. The relationships are therefore restored by applying a local data transformation step. An according data transformation technique will be presented in Chapter 6. Here, it is implemented as a SPARQL construct query. These types of queries allow to build an RDF graph by instantiating a predefined Basic Graph Pattern with the results obtained from executing the query. Data source B is a native RDF dataset without schema information. This information is added in a local annotation step. Afterwards, a global annotation step can be utilized to insert links and mappings between the local datasets. In the example, this is performed for attributes which represent a global patient identifier. Based on these annotations a global data transformation step can be executed in order to materialize equivalences between patient identities from different systems.

Schema Evolution

As shown in the previous example, there are several ways in which schema information can evolve in a system which implements the incremental ontology-based integration approach. Firstly, the RDF data model is inherently schema-free, which allows to integrate datasets that do not provide any schema information. Secondly, the RDF data model allows to incrementally add and refine schema-level descriptions. The most commonly used vocabulary for this purpose is RDF Schema, although there are other vocabularies, such as OWL, which provide richer semantics. In RDF Schema it is, e.g., possible to incrementally define classes of resources and state that a resource is an instance of one or many of these classes. Classes can be structured in a class hierarchy which allows multiple inheritance. An example can be seen in Figure 3.7 where it is defined that the resource *<B:Subject321>* is an instance of the local class *<B:Subject>*. In the same way that a class hierarchy can be built for subjects and objects, a hierarchy of properties can be built for predicates. Additionally, the semantics of predicates can be refined by incrementally defining valid domains and ranges. In Figure 3.7 it is, e.g., stated that the property *<A:patid>* is a subproperty of the global property *<G:globalPatientIdentifier>*.

Schema-relaxed Querying

A dataspace will probably contain highly heterogeneous data, which is not yet integrated or has an unknown or unreliable structure. It is therefore very important to provide means for explorative querying, which can be performed without any prior in-depth knowledge of the queried data. As has already been briefly described in Section 3.2.2, RDF and SPARQL are very well suited for this purpose. For example, it is possible to query for resources (and properties) which have an arbitrary attribute with a given value by including a triple pattern with a variable subject and predicate as well as a constant object:

- `(?x ?y "value"^^<datatype>)`

Additionally, it is possible to retrieve all resources which have an attribute with values of a given data type by including a completely unspecified triple as well as a triple which restricts the range of the variable object:

- `(?x ?y ?z)`
- `(?y <rdfs:range> <datatype>)`

It is also possible to query for resources which have an attribute that is a sub property of a given property:

- `(?x ?y ?z)`
- `(?y <rdfs:subPropertyOf> <property>)`

One can query for resources which are instances of a predefined class:

- `(?x <rdf:type> <Class>)`

It is also possible to query for instances of a subclass of a given class:

- `(?x <rdf:type> ?c)`
- `(?c <rdfs:subClassOf> <Class>)`

Note that, when semantic reasoning is applied within the dataspace, most of the explicit relationships in the example queries will be included automatically. For example, querying for all instances of a certain class, would also return instances of its subclasses. These examples show that there are numerous ways in which a schema-relaxed, explorative querying paradigm is supported by RDF and SPARQL. This flexibility is utilized in many ways by the implementation presented within this thesis. This includes the extraction of implicit knowledge from the global ontology and a browsing component. Moreover, the rich semantics of ontology-languages like RDF/S and OWL are used for annotations and semantic integration.

3.3 Summary and Perspectives

The proposed methodology is related to the *Semantic Web* and *Linked Data* movements. The basic idea of the Semantic Web is to develop a structured web of data as opposed to the unstructured content that currently dominates the World Wide Web (WWW) [BLHL01]. Also building upon the RDF data model, the ultimate goal is to enrich the WWW with structured data in order to create a machine-readable global data space and knowledge base. The Linked Data principle is closely connected to this vision and describes a specific way of publishing RDF data in the WWW [HB11]. It builds upon the following principles: 1) objects are always identified by URIs, 2) only dereferenceable HTTP URIs are used, and 3) when dereferencing a URI, related information for the object (including a list of associated objects) is returned in an RDF format [LIN].

Especially the idea of Linked Data is also driven by the dataspaces paradigm. Here, the hope is that the flexible way of publishing and accessing data accelerates the development of a global Semantic Web. Data publishers are responsible for annotating their data and integrating it with other datasets from the Linked Data Cloud. In this way, the required work is spread amongst the different participants of the dataspace. Similar to our approach, the concept makes heavy use of the ability of RDF to reference arbitrary data items that are distributed over separate databases. In some projects this has also been implemented for the life sciences domain, e.g., in [HKL$^+$09] a linked dataspace has been built with publicly available data about clinical trials.

In comparison to these concepts, the approach proposed in this thesis is less generic and solely oriented towards the requirements of the application domain. It does not aim at building a flexible global dataspace, but to provide powerful means for integrating highly heterogeneous data in a rather local, constantly evolving environment. It also implements a very different data access and querying paradigm, which does not impose any restrictions on the RDF representation of data. In our system, information is managed in distributed RDF databases and we provide highly efficient means for query processing in this environment. This is very hard to achieve for Linked Data due to the underlying data publishing concept. Because of its inherent distribution, the presented techniques are also applicable to some challenges in the Semantic Web context, though. Similar to Linked Data, our approach also aims at carrying out semantic integration in flexible, loosely coupled, robust and demand-oriented manner. But these data integration efforts are supported by an integrated workflow engine which provides common operators and is tightly coupled to the distributed system. This, for example, allows to utilize the computing power of all participants for complex data integration tasks. In contrast to the inherent genericity of the interface provided by Linked Data, we present a tailored graphical user interface and an overall system architecture, which integrates all aspects into a single system with common access control mechanisms.

CHAPTER 4

Laying the Groundwork: Accessing Biomedical Data Sources

The access to biomedical data sources which contain important primary data (e.g., clinical information systems, research databases, biobanks, biomolecular databases) is an important requirement in translational medical research. Most of these systems are implemented on top of relational databases. In patient-care, information systems also exchange information via standardized HL7 message streams. This section presents two components, which allow to transform data from these important types of sources into an RDF representation. This allows to seamlessly integrate biomedical data into the global ontology.

4.1 Related Work[1]

Many solutions for accessing differently structured data from within Semantic Web applications have been proposed. These solutions cover a broad design spectrum, as has been investigated in [GC07]. Here, the authors distinguish between a database- or ontology-centric transformation. A database-centric transformation creates an ontology from the underlying database schema, e.g., by mapping database tables to classes, rows to subjects, columns to attributes and cells to objects. In contrast, an ontology-centric transformation maps a database schema onto an existing local or global ontology. In general, this requires the formulation of much more complex mapping definitions. In [ADL+09] it is further distinguished between manual, semi-automatic and fully-automatic transformation processes, as well as domain-specific and domain-independent approaches. An overview over these key criteria is shown in Figure 4.1. Most previous solutions focus on the widespread relational data model, but approaches for other data models have also been proposed, e.g., XML [Bre09] or Excel Spreadsheets [LW09b].

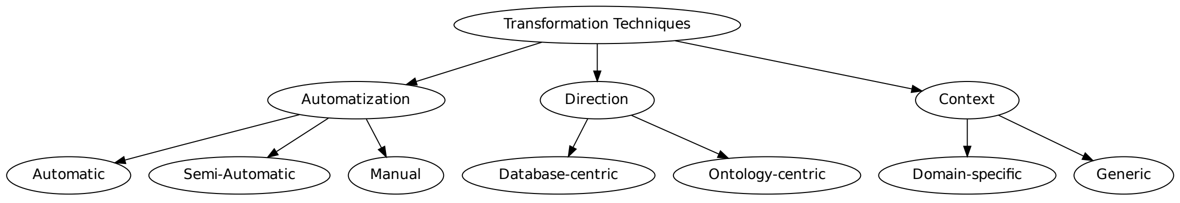


Figure 4.1: Techniques for mapping biomedical databases to RDF

[1]Parts of the work presented in this section are based on the student project [Str11]

Database-Centric Transformation: A database-centric transformation process creates a new ontology from the underlying database schema. Afterwards, an RDF representation of the database is created by exporting the relational data into RDF triples. The resulting ontology closely resembles the original database schema and does therefore not change the semantics and relationships between the data items. Such transformations normally map database tables onto ontology classes, rows onto subjects, columns onto predicates and cells onto objects. Because database-centric transformations normally implement a fully- or at most semi-automatic process, the resulting mappings often lack support for fundamental data transformations.

The approach presented in [GC07] implements the database-centric approach. The mapping creation process builds upon an analysis phase in which existing tables and foreign key constraints are included into the resulting ontology. Depending on the information available in the database's metadata catalog, the tool is able to detect and export one-to-one, one-to-many and many-to-many relationships. Afterwards, the relational data is mapped onto the resulting ontology. To this end, a cell value is transformed to a literal if it is not a foreign key, otherwise it is transformed into a resource.

In [Biz03] an XML-based, declarative mapping language for relational databases and OWL ontologies has been proposed. In order to provide a high degree of flexibility, SQL statements are utilized for the selection of record sets for a certain class or set of classes. Afterwards, the resulting record sets are grouped over a set of defined columns and mapped to RDF instances. In a succeeding step, the instances are generated by assigning URIs or blank node identifiers. Finally, attribute values are attached to the generated resources. This also includes mapping the data types of the underlying database system to XML Schema data types. Due to the incorporation of complex SQL statements, the approach allows to handle different types of relationships and highly normalized data representations. This comes at the costs of a complex and time-consuming mapping process.

Ontology-Centric Transformation: Ontology-centric approaches have been developed for cases in which a database needs to be mapped onto an existing ontology. To this end, the database tables and their attributes need to be related to data items within the target ontology. As this includes complex transformations, it requires error-prone, time-consuming manual work.

In [BCGP04] an extension of [Biz03] has been presented, which allows to map relational schemas to RDF schemas. Graphical tools, such as the one presented in [RGP06], simplify the creation of suitable mapping definitions. The mapping language can deal with cases in which the source and destination schema overlap fully, partially or not at all. The basic functionalities provided include means to map individual tuples from a relation onto several different instances of a single class or different classes. It also supports more complex transformations by incorporating SQL statements.

Materialization or Query Rewriting: Some approaches not only allow to physically transform datasets, but also provide means for on-demand query translation. In the latter case, SPARQL queries against the virtual RDF dataset are rewritten to SQL queries against the relational database system. Both approaches have upsides and downsides. Although query rewriting offers access to an up-to-date database, it puts a significant additional load onto the underlying database system. Additionally, the different data models and their dedicated query languages implement very diverse query processing paradigms. Query translation therefore often suffers from severe performance limitations. This can be circumvented by periodically loading the RDF view into a dedicated RDF database, which comes at the cost of reduced data freshness.

Manual, Semi-Automatic or Automatic Mapping Definition: Transformation solutions can also be categorized by whether they allow for an automatic or semi-automatic mapping definition or require a completely manual process. In contrast to database-centric approaches, ontology-centric solutions always require at least some manual work due to the complexity of the underlying mapping problem.

Semi-automatic processes require much fewer manual steps. These approaches can be divided into two different phases. Firstly, the underlying database schema is analyzed and a preliminary ontology is created. This also involves representing the data items in the relational database as instances of classes in the temporary ontology [LDW05]. Secondly, this model can be refined by the user. To this end, the systems offer interfaces for the user to examine and manually adjust the automatically generated mapping definitions.

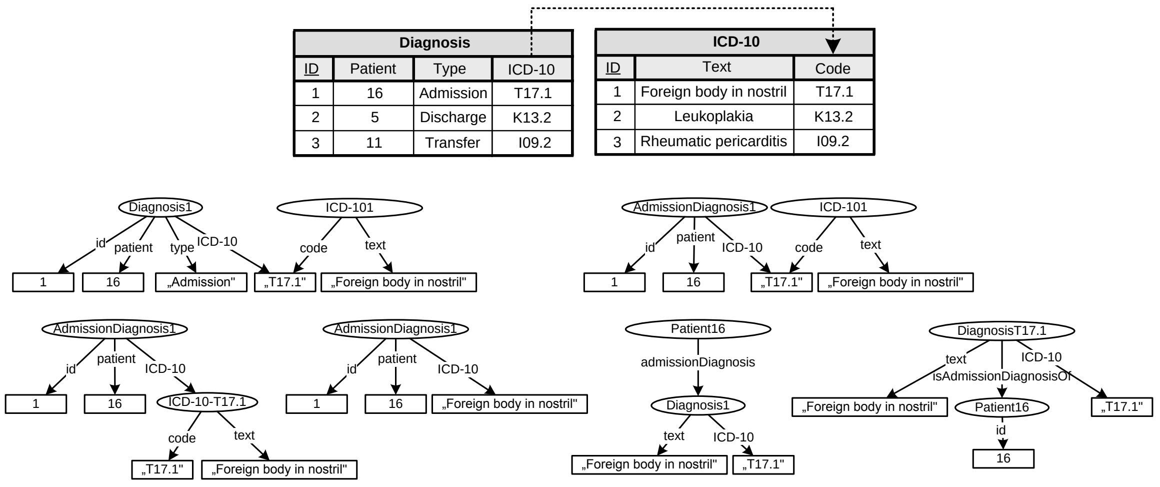
A fully automatic process only requires the user to define which database should be transformed. This type of transformation approaches always follow the table-to-class, row-to-instance, column-to-attribute scheme and try to derive relationships between classes from database constraints stored in the metadata catalogue. This is only feasible for rather simple database schemas without resulting in information loss or inconsistencies.

Domain Dependence: Some approaches are domain-dependent and implement the ontology-centric concept. They are able to take into account existing domain ontologies and integrate them into the mapping definition and data transformation process.

An important type of domain-specific data transformation approach is targeted against HL7 messages. HL7 is a widespread messaging standard for information exchange between clinical information systems [HL7]. It is part of an application-level protocol, which is typically managed via a communication server. This server receives messages from the individual subsystems and implements a selective broadcast mechanism. For translational medical research, HL7 messages contain several important types of data. A lack of metadata and well-defined semantics is a general problem in earlier versions of HL7, as it has been designed to be highly flexible and easily adoptable. In order to overcome these limitations, HL7 V3 has been redeveloped from scratch utilizing a common domain model, the Reference Information Model (RIM). It is strongly object-oriented (designed with UML), makes extensive use of wide-spread terminologies and the data formats (e.g., for messages) are based upon XML. Some previous work have aimed at bringing HL7 V3 to the Semantic Web. In [HCL] and [PRO] RDF representations of important parts of HL7 RIM have been developed. Some work (e.g., [KRA06, GRD]) have utilized XSLT to derive RDF representations of XML-based HL7 V3 messages and integrate them with the HL7 RIM. As there is currently no comprehensive RDF representation of HL7 RIM, these approaches only implement limited example scenarios. Other work (e.g., [JS12]) aim at the opposite direction and utilize RDF to simplify the process of mapping clinical data to HL7 V3 to foster interoperability. Deriving an RDF representation is rather simple for HL7 V3 messages, as these are XML-based and contain meaningful metadata. Unfortunately, HL7 V3 is not compatible to previous versions. As a migration is highly complex, HL7 V3 is only rarely used whereas earlier versions (especially HL7 V2) are widely deployed. Section 4.4 focusses on deriving an RDF representation from HL7 V2 messages.

4.2 Challenges and Requirements

This section presents basic requirements for the transformation of biomedical data sources into an RDF representation. To this end, it utilizes some exemplary scenarios for the transformation of relational data. Figure 4.2 shows an example dataset which associates diagnoses to patients. A table *Diagnosis* stores patient identifiers and associated diagnose identifiers. Additionally, it is stored whether the diagnosis has been made at admission, transfer or discharge. The diagnoses themselves are encoded as ICD-10 disease codes. A second table, *ICD-10*, stores a textual description of each diagnosis code.



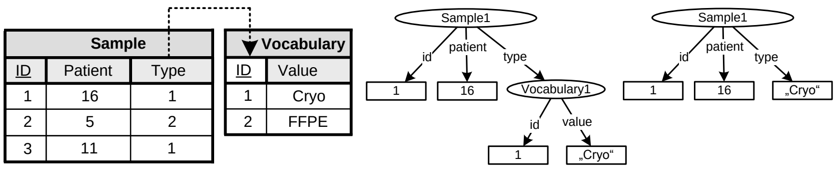
Diagnosis			
ID	Patient	Type	ICD-10
1	16	Admission	T17.1
2	5	Discharge	K13.2
3	11	Transfer	I09.2

ICD-10		
ID	Text	Code
1	Foreign body in nostril	T17.1
2	Leukoplakia	K13.2
3	Rheumatic pericarditis	I09.2

Figure 4.2: Relational data and RDF representations

Below the relational data, the figure shows six possible RDF representations of the first entry in the Diagnosis table as well as the associated ICD-10 description. In clockwise direction, the first RDF representation shows a simple transformation following the table-to-class, row-to-instance, column-to-attribute and cell-to-value (TRCC) model. There are several potentially reasonable representations of these data items which do not strictly follow the TRCC model. Firstly, the semantics of the underlying model might be captured better by defining different types of diagnosis classes, such as *Admission Diagnosis*, *Transfer Diagnosis* or *Discharge Diagnosis*. Secondly, the relationship from the diagnosis to the according ICD-10 metadata is only represented implicitly (both subjects are linked to the same object). It might be desirable to explicitly define this relationship by including a triple which reaches from one resource to the other. In some cases it could even make sense to ignore the second resource and only reference the description of an ICD-10 code. Other potentially meaningful transformations include replacing the literal representation of the patient identifier by a resource which represents the patient. This resource could then be connected to the diagnosis code by either inserting a triple reaching from the patient to the diagnosis, or from the diagnosis to the patient.

Figure 4.3 shows an example for a generic relational database schema. Related designs are often used by Electronic Data Capture (EDC) systems, which for example implement a generic EAV schema. These systems offers means to dynamically specify the vocabulary which is used for data entry. To this end, terminological control is imposed by referencing (user- or administrator-defined) vocabularies from within the database tables. In the example, the system manages biospecimens and stores metadata about how the samples have been preserved. This information is encoded in an associated data dictionary. As can be seen in the proposed RDF representations, a meaningful transformation of these data items might completely leave out the *Vocabulary* relation.

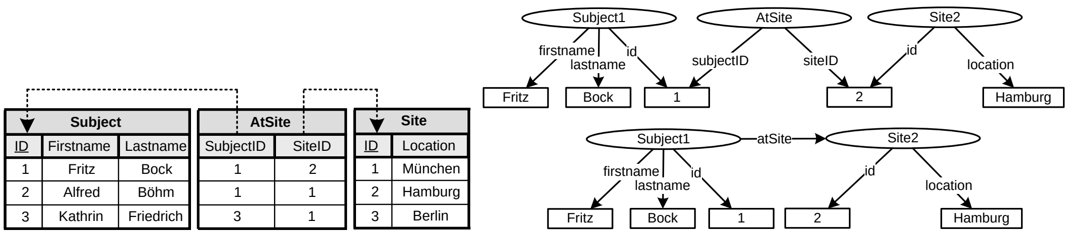


Sample		
ID	Patient	Type
1	16	1
2	5	2
3	11	1

Vocabulary	
ID	Value
1	Cryo
2	FFPE

Figure 4.3: Generic relational data and RDF representations

Figure 4.4 shows a scenario which involves the transformation of a many-to-many relationship. Analogously to the other two scenarios, a useful transformation beyond the simple TRCC model would be to explicitly define these relationships. This means that the information stored in the table *AtSite* is only implicitly contained in the resulting dataset.



Subject		
ID	Firstname	Lastname
1	Fritz	Bock
2	Alfred	Böhm
3	Kathrin	Friedrich

AtSite	
SubjectID	SiteID
1	2
1	1
3	1

Site	
ID	Location
1	München
2	Hamburg
3	Berlin

Figure 4.4: Relational many-to-many relationship and RDF representations

The general idea behind the concept of incremental ontology-based integration is to perform a generic transformation of biomedical data sources into an RDF representation. On the other hand, there are various transformations which would be highly useful in real-world scenarios. This includes representing data only implicitly or modifying the class hierarchy which would be generated by the highly generic TRCC model. The requirements of simplicity and genericity as well as the ability to apply complex transformations are at opposite ends of the design space. This section therefore presents different approaches for different data models which are generic but allow for a semi-automatic process that supports reasonable transformations. If the presented extraction process is not able to completely cover the semantics of the underlying model, the data transformation techniques presented in Section 6 can be applied to post-process the results accordingly.

4.3 Relational DBMS [2]

In this work, the relational model is the most important data model, because relational database systems are the dominating storage solution within the biomedical domain. In conformance to the requirements outlined previously, this section proposes a highly

[2]Parts of the work presented in this section are based on the student projects [Str11], [Tro11] and [Vaa11]

flexible, yet easy to use and generic solution for this type of data. The key idea is to provide the ability to define a directed acyclic graph (DAG) of operators which can be applied to RDF data items. The bridge between the RDF world and the relational world is implemented with a special operator (*produce*) which generates RDF representations of tuples from a relation following the TRCC model. In contrast to previous work, there is not need to specify SQL statements or complex mapping rules. Although it is of course possible to define complex transformations with the operators presented, this process is supported in numerous ways:

- The data modification operators solely operate on RDF data, which is also the underlying model of the whole integration system itself. The users are not required to have any in-depth knowledge about relational databases or SQL. The *produce* operator which transforms relational data into RDF data items has well-defined, easy to understand semantics.

- Building the DAG of transformation operators is fully supported by a graphical user interface. One of the key ideas behind the approach is that the operators are designed in a way which allows for explicit immediate visual feedback at each step in the process.

- Additionally, a semi-automatic process is implemented that automatically proposes an initial mapping by accessing the metadata catalog of the database management system.

4.3.1 Transformation Operators

There are five different operators which form the backbone of the transformation process. Despite the *produce* operator which does not have a child operator, the binary and unary operators can be arranged in a directed acyclic graph. The operators process sets of *RDF data items*. Upon evaluation, each operator also returns a set of RDF data items. An RDF data item is basically an arbitrary RDF graph following a predefined schema.

- The nullary PRODUCE operator returns an RDF representation of the tuples of a database table. It defines a class for the instances of the table and generates RDF data items following the TRCC model.

- The unary PROJECT operator removes properties from all instances of a predefined class or alters the definition of their URIs.

- The unary FILTER operator removes all instances of a predefined class which do not conform to a specified expression.

- The binary RELATE operator inserts relationships between instances which share common values for predefined attributes.

- The binary MERGE operator merges instances which share common values for predefined attributes.

The operators can be organized in a directed acyclic graph which consists of distinct, connected subgraphs that define *fragments*. The RDF representation of the relational data is defined as the union over the results of all fragments. The DAG structure allows to reference (i.e., re-use) operators which have been defined in other fragments which avoids duplicate work. For example, the filter operator might be used to only transform some tuples stored in a relation (e.g., due to legal reasons). In order to prevent information leakage it is recommended in this case to explicitly define a single fragment which only transforms these tuples. This fragment can then be re-used in any other fragment which references the resulting entities.

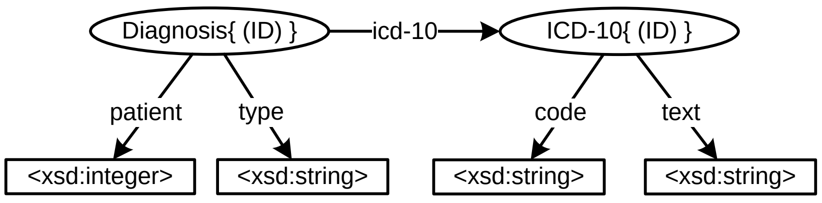


Figure 4.5: A set of RDF data items

Preliminaries: The operators presented in this section work on so called *RDF data items*. These consists of *RDF entities* and *RDF relationships*.

Definition 8 (*RDF Data Item*)

A set of RDF data items I is defined as $E \cup R$ where

1. *E is a set of RDF entities, and*
2. *R is a set of RDF relationships.*

An RDF entity defines a schema for a set of RDF graphs which describe similar entities. An entity is defined by a resource which is a subject in several triples which have literal objects. In the context of RDF entities, each of these triples is called a property and has a predefined name.

Definition 9 (*RDF Entity*)

An RDF entity e defines a schema for a set of RDF graphs. It is defined as a tuple $\mathsf{e} = (\mathsf{name}, \mathsf{I}, \mathsf{A})$ where name is a string which defines the resources' names, I is a set of tuples containing attribute names and A is a set of properties. A property $\mathsf{a} \in \mathsf{A}$ is a tuple $\mathsf{a} = (\mathsf{name}, \mathsf{xsd})$ of a string name that encodes the property's name and the URI xsd of a an associated XML Schema data type.

As can be seen from this definition, RDF entities only have properties with literal values. RDF relationships provide further means to model relationships between the entities themselves, i.e., to include triples which have resources as subject and object.

Definition 10 (*RDF Relationship*)

An RDF relationship r defines a relationship between two RDF entities. It is defined as a tuple $\mathsf{r} = (\mathsf{subject}, \mathsf{name}, \mathsf{object})$ where $\mathsf{subject}$ and object are RDF entities and name is a string which labels the relationship.

An RDF representation of the relational schema from 4.2 is shown in Figure 4.5. It contains two RDF entities (Diagnosis and ICD-10) and one RDF relationship (icd-10) which relates the entities. Each of the entities' attributes has an associated XML Schema data type.

Produce: Given these definitions we are now able to define the transformation operators. We start by defining the *produce* operator which is acts as a bridge between the relational world and the RDF world by returning an RDF entity for a given relation.

Definition 11 (*Produce operator*)

A produce operator Produce *is a tuple* (r, n, I, A) *where* r *is the name of a relation in the database,* n *is the name of the resulting RDF entity,* I *is a non-empty totally ordered set of attributes of the relation and* A *is a potentially empty set of attributes of the relation.* I *and* A *may overlap.*

Upon evaluation, a produce operator returns an RDF entity which is derived from the underlying relation r and the sets I and A. The entity is named n and the data types of the entitie's properties are automatically derived from the SQL data types of the corresponding attributes. To this end, a mapping table is utilized which defines a mapping between SQL data types and XML Schema data types. An example mapping for a subset of the SQL data types is shown in Table 4.1.

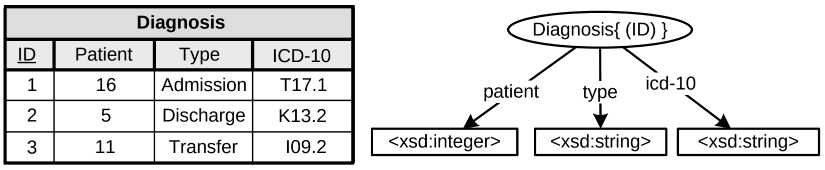
SQL	XSD
VARCHAR	string
INTEGER	integer
DOUBLE	double
DATE	date
TIME	time

Table 4.1: Mapping SQL to XML Schema data types

It is important to note that the ability to define an arbitrary name for the resulting entity allows to generate different entities from the same underlying database table. This is useful in many ways, e.g. when two different entities which cover different subsets of the attributes are to be generated from the same relation. The RDF entity $entity = (n, I, A)$ which is returned by a produce operator $produce = (r, n, I, A)$ is defined as:

- $entity.n = produce.n$,
- $entity.I = \{produce.I\}$,
- $entity.A = \{a = (n, r) | n \in produce.A \wedge r = map(datatype(n))\}$.

An example definition of a produce operator for the table *Diagnosis* from Figure 4.2 is shown in Figure 4.6. The operator which can be applied to derive the presented RDF entity is:
$produce = ("Diagnosis", "Diagnosis", ("ID"), \{"patient", "type", "icd-10"\})$.



Diagnosis			
ID	Patient	Type	ICD-10
1	16	Admission	T17.1
2	5	Discharge	K13.2
3	11	Transfer	I09.2

Figure 4.6: Example for the produce operator

Given the definition of the produce operator, it is now possible to present the transformation operators. Each of these operators consumes one or two sets of connected RDF data items and returns a set of connected RDF data items. Connected means that the resulting directed graph structure (as, e.g., shown in Figure 4.6) is weakly connected.

Project and Filter: The project and filter operators manipulate a connected set of RDF data items. The project operator allows to change an entity's schema, whereas the filter operator can be used to drop instances which do not fulfil a given expression.

Definition 12 (*Project operator*)

A project operator Project *is a tuple* (R, e, I', A') *where* R *is a connected set of RDF data items,* e ∈ R *is an RDF entity and* I' *is a set of tuples containing attribute names and* A' *is a set containing attribute names. All attributes in* I' *or* A' *must also be contained in either* e.A *or in* e.I*.*

The result of the application of a *project* operator (R, e, I', A') on a set of RDF data items R is a new set of RDF data items in which the entity $e = (n, I, A)$ is replaced by a new entity $e' = (n, I', A')$. A project operator also allows to exchange an entity's attributes and identifiers. This means that, e.g., a part of an identifier can become an attribute or vice versa.

The filter operator can be used to drop all instances of the pattern described by the set of RDF data items that do not fulfil a given predicate.

Definition 13 (*Filter operator*)

A filter operator Filter *is a tuple* (R, p) *where* R *is a connected set of RDF data items and* p *is a predicate which is build upon the names of identifiers and attributes of any RDF entity* e ∈ R*.*

This means that the filter operator does not have any effect on the actual RDF data items it processes. Therefore an operator (R, p) which operates R also returns R but only has an effect during the execution of the transformation process. For example, the operator $filter = (R, (type = \text{"}Admission\text{"}))$ could be utilized to extract all admission diagnoses from the result the produce operator which produces the RDF entity that is shown in Figure 4.6.

Relate and Merge: Previous operators produce or manipulate sets of RDF entities. The *relate* operator allows to define links between RDF entities and thus to insert RDF relationships into a set of RDF data items. The *merge* operator works analogously but merges two related entities into a single entity.

Definition 14 (*Relate operator*)

A relate operator Relate *is a tuple* ($name$, R_1, e_1, a_1, R_2, e_2, a_2) *where* name *is the label of the generated relationship,* R_1 *and* R_2 *are two different connected sets of RDF data items and* e_i *is an entity in* R_i *and* a_i *is an attribute of* e_i*.*

An operator Relate=($name$, R_1, e_1, a_1, R_2, e_2, a_2) inserts a relationship between all instances of e_1 and e_2 with equal values for the attributes a_1 and a_2. Therefore, on schema level, this operator returns a connected set of RDF entities $R = R_1 \cup R_2 \cup \{(e_1, name, e_2)\}$. On instance level, this relationship is only inserted for pairs of instances (e_1, e_2) with $e_1.a_1 = e_2.a_2$.

Definition 15 (*Merge operator*)

A merge operator Merge *is a specialization of a relate operator which is defined analogously but has different semantics.*

An operator Merge=($name$, R_1, e_1, a_1, R_2, e_2, a_2) does not insert a relationship, but merges all instances e_1 and e_2 with equal values for the attributes a_1 and a_2 into a common instance of the class *name*. On instance-level this merge is performed for pairs of instances (e_1, e_2) with $e_1.a_1 = e_2.a_2$. The definition of the result on schema-level is more complex. The two input sets R_i are processed independently and the entities $e_i = \{name_i, I_i, A_i\}$ are replaced with a new entity $merged = \{name, I_i, A_i\}$. Moreover, all relationships are adjusted accordingly. The set $Relationships_i = \{(e_1^{'}, name^{'}, e_2^{'}) \mid e_1^{'} = e_i \vee e_2^{'} = e_i\}$ contains all relationships in R_i that range from or to e_i. In detail, the following is performed:

1. The original entity is removed: $R_i = R_i \setminus \{e_i\}$.
2. The original entity's relationships are removed: $R_i = R_i \setminus Relationships_i$.
3. The new entity is added: $R_i = R_i \cup \{merged\}$.
4. The outgoing relationships of the new entity are added:
   $R_i = R_i \cup \{(merged, name^{'}, e_2^{'}) | (e_i, name^{'}, e_2^{'}) \in Relationships_i\}$.
5. The incoming relationships of the new entity are added:
   $R_i = R_i \cup \{(e_1^{'}, name^{'}, merged) | (e_1^{'}, name^{'}, e_i) \in Relationships_i\}$.

After this process has been applied to both input sets R_1 and R_2, the result of applying the Merge operator is defined as $merge(R_1, R_2)$. The function $merge(R_1, R_2)$ merges two sets of RDF data items and is of general importance because it is also utilized to combine the results of the individual fragments.

Definition 16 (*Merge function*)

The function merge(R_1, R_2) merges two sets of RDF data items R_1 and R_2. It returns a new set R in which all entities from R_1 and R_2 which have the same name are merged. To this end, a new entity is created whose set of attributes and primary keys is defined as the union of the two sets of attributes from the entities in the input sets. All RDF relationships are altered accordingly.

4.3.2 Transformation Process

The transformation process can be executed as soon as a set of operators has been specified which define the transformation process. It is important to note that by defining these operators a data integrator has not only defined the structure of the resulting RDF data, but also how these data are to be generated from the underlying database. To this end, the DAG of transformation operators is utilized to fully automatically generate a set of SQL queries. The following example will be used throughout the remainder of this section. It generates the RDF representation shown in Figure 4.5 from a relational database following the schema from Figure 4.2:

```
project( relate( produce("ICD-10", "ICD-10", ("ID"), {"code", "text"}),
                 "ICD-10", "code",
                 produce("Diagnosis", "Diagnosis", ("ID"), {"patient","type","icd-10"}),
                 "Diagnosis", "icd-10"),
         "Diagnosis, ("ID"), ("patient"), ("type") ) )
```

The first of the two **produce**-operators generates RDF entities which represent the ICD-10 diagnosis codes. It generates entities with a class name *ICD-10* whose primary key is defined by the attribute *ID*. The entities further have two attributes, *code* and *text*. The second **produce**-operator generates entities with class name *Diagnosis*, primary key *ID* and attributes, *patient*, *type* and *icd-10*. The **relate**-operator inserts a relationship between instances of these classes if the attribute *code* matches the attribute *icd-10*. Finally, the **project**-operator is utilized to drop the redundant attribute *icd-10* from the instances of the class *Diagnosis*.

This operator-tree specifies a single fragment which defines several different aspects of the transformation process. Firstly, it defines *what* the schema of the resulting RDF graph looks like, although the actual export vocabulary can still be redefined. Secondly, it also defines *how* the data from the underlying database system needs to be transformed in order to result in the desired export schema. To this end, the transformation operators are substituted by SQL expressions:

- The **produce** operator defines the name of the underlying relation and the attributes to be transformed. When compiled into a SQL query it defines a unique identifier which is used to reference tuples that represent instances of the according entity. In the same way as the **project** operator the produce operator also influences the *SELECT* clause of the resulting SQL query.
- The **filter** operator defines a predicate which is compiled into the *WHERE* clause of the query.
- The **project** operator does only affect the resulting RDF data items which is reflected in the *SELECT* clause of the resulting SQL query.
- The **relate** and **merge** operators are compiled into *INNER JOIN*s over the relations defined by the produce operators.

According to this process, the SQL query for the example fragment is defined as:

```
SELECT d1.id, d1.patient, d1.type, i1.id, i1.code, i1.text
FROM Diagnosis AS d1 INNER JOIN ICD-10 AS i1
ON d1.icd-10=i1.code
```

Executing this query would result in a set of tuples with the following schema:

```
(Diagnosis.id, Diagnosis.patient, Diagnosis.type, ICD-10.id, ICD-10.code, ICD-10.text)
```

The resulting RDF data is then generated by production rules which read the queries' result sets on a tuple per tuple basis and generate triples whenever all input attributes of a rule are not *NULL*. The set of production rules for the example transformation is:

```
// Entity:  "Diagnosis"
(d1.id, d1.patient) → (Diagnosis[d1.id], patient, convert(d1.patient, <xsd:integer>))
(d1.id, d1.type)    → (Diagnosis[d1.id], type, convert(d1.type, <xsd:string>))
// Relationship
(d1.id, i1.id)      → (Diagnosis[d1.id], icd-10, ICD-10[i1.id])
```

```
// Entity:  "ICD-10"
(i1.id, i1.code)   → (ICD-10[i1.id], code, convert(i1.code, <xsd:string>))
(i1.id, i1.text)   → (ICD-10[i1.id], text, convert(i1.text, <xsd:string>))
```

In this example, the square brackets indicate that the contained attribute (including the brackets) is to be replaced by the value defined by a resulting tuple in order to generate an instance identifier. The function *convert* generates a valid RDF literal for each binding to the variable and the defined type, e.g. *convert("T17.1", "<xsd:string>")* returns `"T17.1"^^<xsd:string>`. In addition to these rules, which instantiate RDF entities as well as their relationships, additional triples are generated to link the instances to an automatically generated RDF/S schema description. For the example, the following two additional rules would link each instance of an RDF entity to its according RDF/S class definition:

```
(d1.id) → (Diagnosis[d1.id], type, Diagnosis))
(i1.id) → (ICD-10[i1.id], type, ICD-10))
```

As can be seen, the example transformation results in two class definitions (*Diagnosis* and *ICD-10*). The way in which the according RDF/S description is generated from the resulting RDF data items will be explained in more detail in the following paragraph. In the example, all resource identifiers have been abbreviated by leaving out the prefixes. The production rules also provide means to redefine the vocabulary of the resulting RDF graph. To this end, the graphical tool (see Section 7) allows to rename all entities, attributes as well as relationships and to define the associated namespaces.

The overall process described in this section is shown in Figure 4.7. It can be seen that the metadata (i.e. the schema) of the underlying database forms the input for the produce operators in the fragments. For each of these fragments, a SQL query ($Frag_1$, $Frag_2$ and $Frag_3$) can be generated by compiling the subgraph defined by the entry point. In addition, a set of RDF data items can be generated by applying all operators to the database schema. It is important to note that a SQL query as well as a set of associated RDF data items can be derived from every node in the DAG. This is leveraged by the graphical tool to provide instant visual feedback for each step in the transformation definition process. The production rules form the glue between the RDF data items and the SQL queries and utilize the information from both representations to generate RDF data of the relational data in the database. The RDF/S representation of the resulting RDF data can be generated be interpreting the resulting RDF data items accordingly. To this end, entities are transformed according to the following rule:

Definition 17 (*RDF/S representation of an RDF Entity*)

For an RDF entity e = (Entity, I, A) the following triple is generated:
(Entity, type, Class)
For each Attribute (name, xsd) ∈ A the following triples are generated:
(name, type, Property)
(name, domain, Entity)
(name, range, xsd)

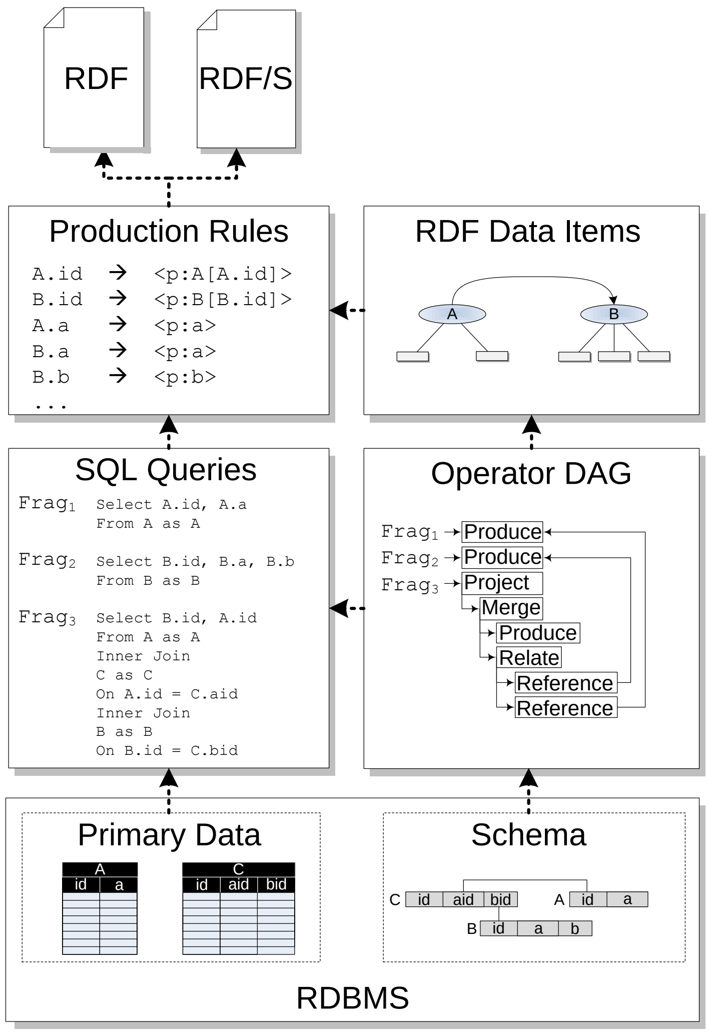


Figure 4.7: Transforming relational data

The RDF/S representation of RDF relationships is defined as follows:

Definition 18 (*RDF/S representation of an RDF Relationship*)

For an RDF relationship r = (Subject, name, Object) the following triples are generated:
(name, type, Property)
(name, domain, Subject)
(name, range, Object)

The generated RDF/S schema definition does exactly describe the schema of the resulting RDF graph and is referenced by each instance generated by the transformation process. In the current implementation, the relate and merge operators are limited to inner joins. Although it would be possible to also incorporate outer joins, the resulting semantics are rather complex. With inner joins, each operator returns a set of connected RDF data items and every instantiation of the items will also result in a complete and connected graph. When allowing outer joins, parts of the resulting instances can have *null* values (as introduced by null values which result from executing the underlying SQL query). Although this allows to apply more complex transformations (e.g., to merge entities only if there is another matching entity and keep the original entity if there isn't), we currently do not allow outer joins. The RDF/S definition which would result from applying the example transformation operators to the schema shown in Figure 4.2 is:

```
// Entity: Diagnosis
(Diagnosis, type, Class)
(patient, type, Property) (patient, domain, Diagnosis) (patient, range, Integer)
(type, type, Property) (type, domain, Diagnosis) (type, range, String)
// Entity: ICD-10
(ICD-10, type, Class)
(code, type, Property) (code, domain, ICD-10) (code, range, String)
(text, type, Property) (text, domain, ICD-10) (text, range, String)
// Relationship: icd-10
(icd-10, type, Property) (icd-10, domain, Diagnosis) (icd-10, range, ICD-10)
```

Applying the transformation to the relation from Figure 4.2 would result in the following RDF instance data:

```
(Diagnosis1, type, Diagnosis) (Diagnosis1, patient, "16") (Diagnosis1, type, "Admission")
(Diagnosis1, icd-10, ICD-101)
(ICD-101, text, "Foreign body in nostril") (ICD-101, code, "T17.1")
(Diagnosis2, type, Diagnosis) (Diagnosis2, patient, "5") (Diagnosis2, type, "Discharge")
(Diagnosis2, icd-10, ICD-102)
(ICD-102, text, "Leukoplakia") (ICD-102, code, "K13.2")
(Diagnosis3, type, Diagnosis) (Diagnosis3, patient, "11") (Diagnosis3, type, "Transfer")
(Diagnosis3, icd-10, ICD-103)
(ICD-103, text, "Rheumatic pericarditis") (ICD-103, code, "I09.2")
```

4.3.3 Evaluation

The evaluation utilizes the *Lehigh University BenchMark (LUBM)* [GPH04, GPH05]. Although this benchmark does not implement a biomedical scenario, it offers ways to repeatedly generate datasets of different sizes which implement a realistic scenario. It is well known and has been utilized for benchmarking many different types of RDF storage solutions (e.g., [HAR11]). LUBM comes with a data generator which is able to create differently sized datasets. This allows to evaluate the scalability of our approach for increasingly complex datasets with the same database schema. The datasets implement an academic scenario which consists of *universities*, *professors*, *students*, *publications* etc. and the relationships between these entities. We have extended the LUBM dataset generator by a component, which is able to generate a relational representation. The schema of the resulting databases is shown in Figure 4.8.

The relational representation is different from the RDF representation in several ways. Although it is being used for different purposes as well, the LUBM benchmark was initially developed for benchmarking RDF reasoners. It therefore generates entities following a comprehensive class-hierarchy, such as the class *Professor* and its subclasses *AssociateProfessor*, *AssistantProfessor* and *FullProfessor*. A similar hierarchy exists for the class *Student* and *Course*. The attributes generated for instances of these subclasses differ slightly. For example, an undergraduate student does not yet have an undergraduate degree. The instances of the different subclasses are also subject to restrictions on their relationships. For example, *UndergraduateStudents* can not take *GraduateCourses*. In the relational representation these subclasses are merged into tables for their common superclass, i.e., *Student*, *Professor* and *Course*. The actual definition of the concrete subclass is achieved by maintaining an additional attribute *type* which can be utilized to distinguish between subtypes. Because of this inherent complexity, a meaningful transformation into RDF includes many of the challenges presented in Section 4.2. For example, the entities *Professor* and *Student* need to be transformed into different specializations (e.g. *GraduateStudent* and *UndergraduateStudent*) depending on the value of certain attributes. Moreover the auxiliary tables *TakesCourse*, *AuthorStudent* and *AuthorProfessor* need to be transformed into relationships between the entities.

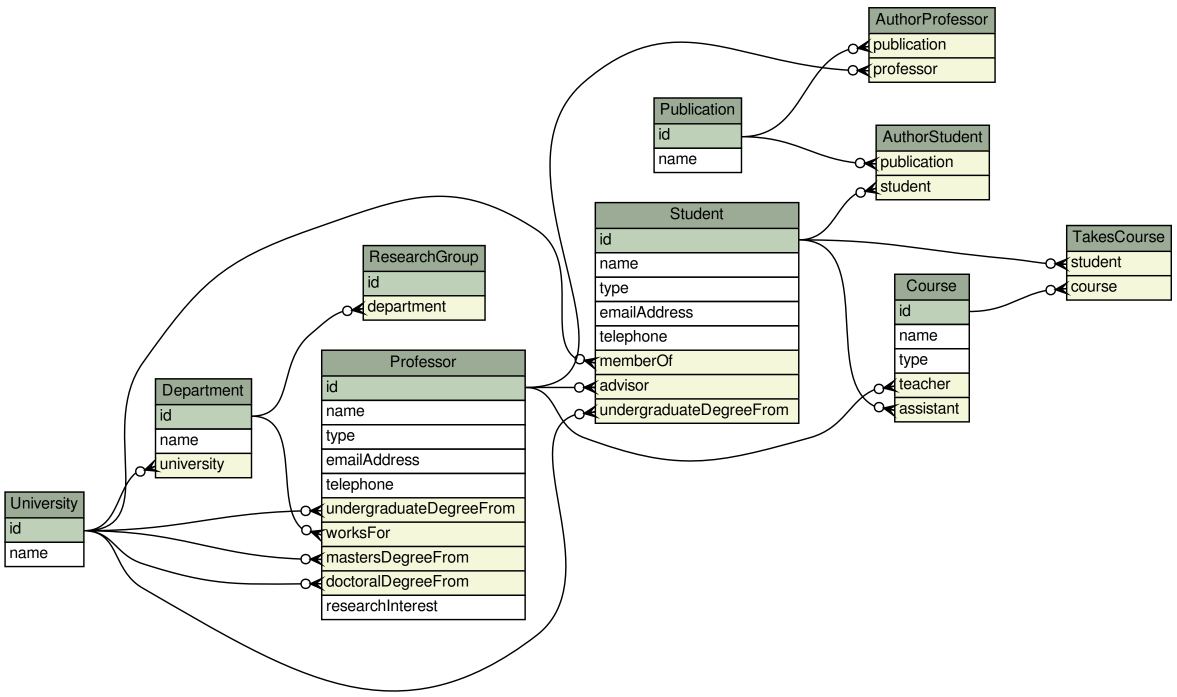


Figure 4.8: Relational schema for the LUBM dataset

An overview over the entities and relationships in the resulting RDF representation is shown in Figure 4.9. The entities' attributes have been omitted for better readability. It can be seen that the resulting RDF dataset contains 11 entities, whereas the original relational schema contained only 7 relations. Furthermore, all entities extend a specific subclass such as *GraduateCourse* and *UndergraduateCourse*. The mapping definition consists of 43 fragments with an average of about 6 operators per fragment. The creation of the mapping definition was carried out with the graphical tool, which will be presented in more detail in Chapter 7.

We created four relational instances of the LUBM datasets with scale factors of 5, 10, 15 and 20. The total number of tuples in each dataset was ∼274.000 for a scale factor of 5, ∼693.000 for a scale factor of 10, ∼990.000 for a scale factor of 15 and ∼1.298.000 for a scale factor of 20. In order to systematically evaluate the scalability of our approach we executed the transformation process for each of these datasets. The experiments were performed on a Dell laptop with a 4-core 1.6 GHz Intel Core i7 CPU with 6 MB cache and 4 GB of memory running a 64-bit Linux 2.6.35 kernel.

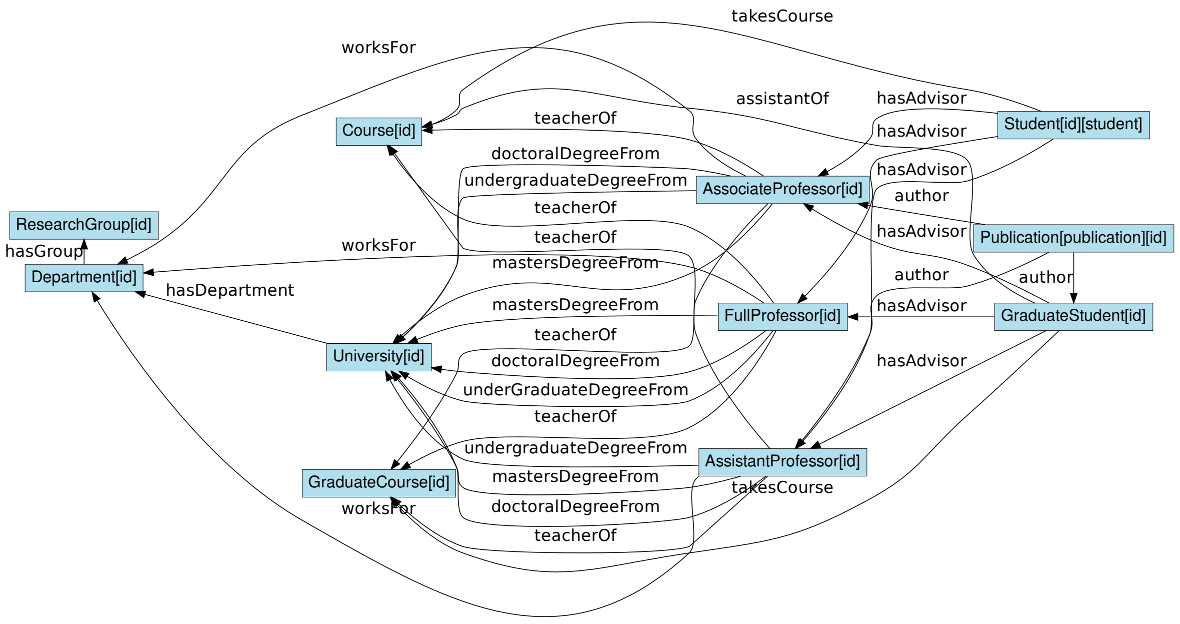


Figure 4.9: RDF schema for the LUBM dataset

The system is able to perform sequential reads and writes on the local hard disks with about 100 MB/s. The data was exported from a MySQL database in version 14.14. All primary keys as well as foreign keys were indexed accordingly. The data from the relational database system is piped through the transformation component, which therefore only has a very small memory footprint.

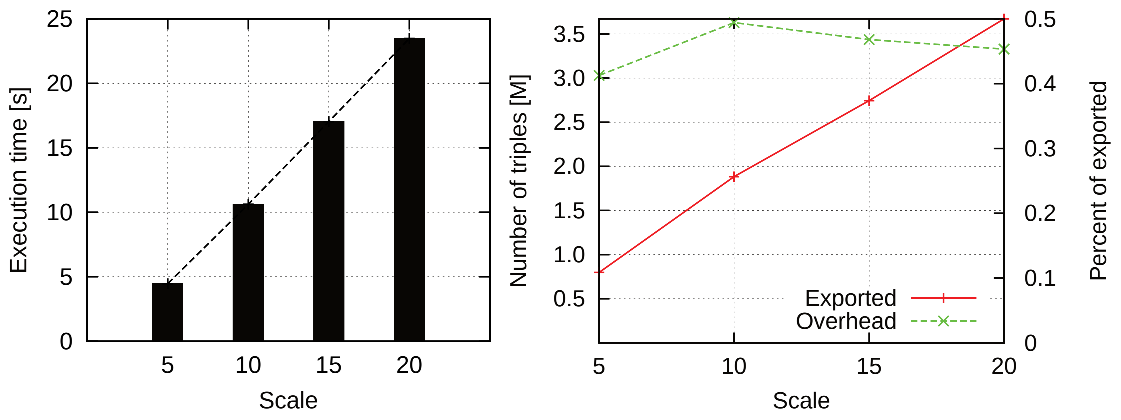


Figure 4.10: (1) Execution times and (2) result sizes for the LUBM dataset

An overview over the resulting execution times is shown in Figure 4.10. It can be seen that the process scales linearly from an initial 3 seconds for the scale factor of 5, to 15 seconds for a scale factor of 20. This shows, that the described data extraction process is well suited for large datasets and complex mapping definitions. The size of the resulting dataset and the overhead induced by the extraction process is also shown in Figure 4.10. The number of exported triples follows the same linear growth as the overall execution time.

The indicated *overhead* gives insights into a side-effect of the presented transformation solution. The problem is that metadata (e.g., the references to the according RDF/S class definitions) needs to be generated for each exported RDF resource. As the same resources are utilized in different fragments, it is not easily possible to determine whether this metadata has already been generated for a resource. Although this could be implemented by maintaining (potentially large) in-memory data structures, we chose a different approach. If a data integrator is willing to reduce this overhead, the transformation component provides means to export these metadata only for predefined fragments. It is important to note that this definition has to be carried out carefully because otherwise the resulting dataset could lack some schema definitions. If the according information is not provided, the transformation component will simply generate redundant triples. Despite the introduced overhead, this is not a problem as these triples will be dropped by any RDF database system when importing the data.

4.4 HL7 Message Streams[3]

HL7 is a widespread messaging standard for information exchange between clinical information systems [HL7]. It is part of an application-level protocol which is typically managed via a communication server. This server receives messages from individual subsystems and passes them on to all systems that have been registered to receive messages of the according type. The communication server thus implements an interface between the systems via a selective broadcast mechanism.

For translational medical research, HL7 messages contain several important types of data. Firstly, it can be easier to extract clinical data (e.g., laboratory data) from HL7 messages than it is to directly access the originating information system (e.g., Laboratory Information System (LIS)). Secondly, administrative information often contains important metadata. An example are reconciliation events, which occur when two different identifiers have been found for the same patient. In this case, the according patient identifiers are reconciled and HL7 messages are sent. Clean identifying data is highly relevant for data integration systems, as it is often required to ensure consistency in replicated data. Even if it is not necessary to alter any data, the availability of such information might be important prospectively.

```
MSH|^~\&|SENDER|099|RECEIVER||20090618122708||ADT^A01|42513321|P|2.3
EVN|A01
PID|||00048441934||Prasser^Fabian^Herr|Prasser|29533212|M|||^^^^^D||||D|||||||||||D|||00000000000000
NK1||Prasser||^^^^^D
PV1||||||||||||||||||||||||||||||||||||||||||||||^^^^^^^^^^^^^^^^^^^^NP0100
```

Figure 4.11: Example HL7 V2 message of type ADT-A01

HL7 defines different message groups each of which again contains different message types. Each message itself is defined by a set of segments which again consist of different groups of fields. There is, e.g., a message group for events regarding the admission, discharge and transfer (ADT) of patients. If a patient is admitted a message of type ADT-A01 is sent to all subsystems in order to inform them about the admission. In HL7 V2 messages are encoded in plain text and separator characters are utilized to encode segments, groups and fields. An example is shown in Figure 4.11.

4.4.1 Transformation Process

This section presents a *fully-automatic*, *domain-dependent* transformation approach for HL7 messages, which implements *materialization.* The approach is able to automatically transform valid HL7 messages into an RDF representation. It is domain-

[3]Parts of the work presented in this section are based on the student project [Sch09]

dependent and implements a concept in-between the design space of *database-centric* and *ontology-centric* solutions. The resulting RDF graph is materialized in a dedicated RDF database system. The resulting system architecture is shown in Figure 4.12. Here, the *HL7 Transformer* is registered as a receiver at the communication server. It transforms all incoming messages into an RDF representation and incrementally maintains a definition of the resulting schema by utilizing the RDF Schema Description Vocabulary (RDF/S).

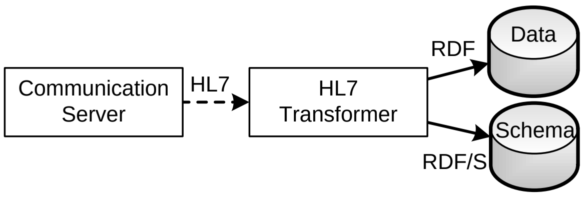


Figure 4.12: Architecture of the HL7 to RDF transformer

To this end each message is automatically transformed into an RDF representation. As HL7 messages in version 2.3. do not contain any metadata such as field names, we utilize a machine-readable representation of the HL7 specification. Such a machine-readable specification is provided by the HAPI project [HAP] which implements a generic Java-based parser for HL7 messages. This parser is automatically generated from the HL7 specification. We use HAPI to parse each message and then utilize Java reflection to traverse the resulting object model which is equipped with meaningful metadata such as field names. An excerpt of the RDF representation of the HL7 message from Figure 4.11 is shown in Figure 4.13.

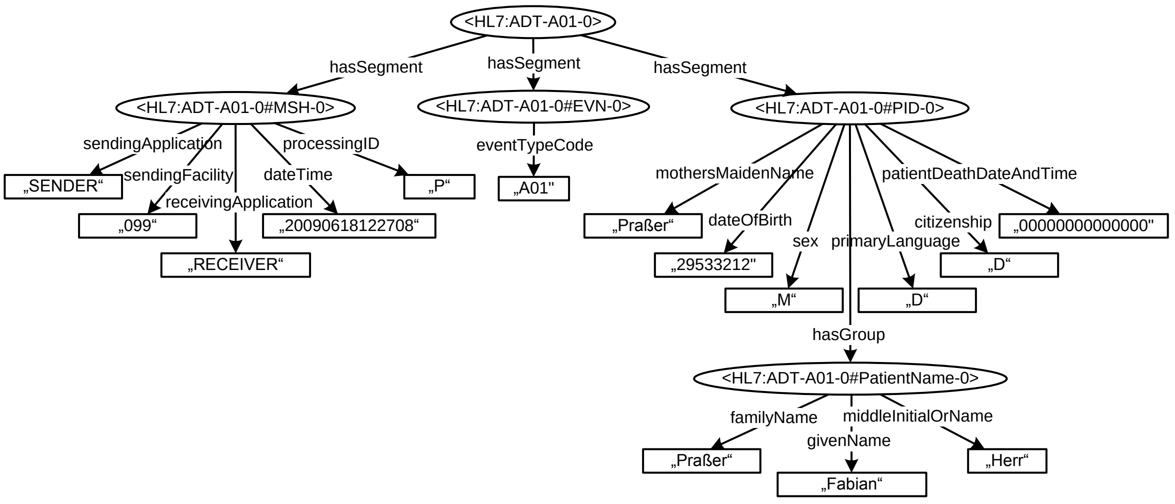


Figure 4.13: Excerpt of the representation of the message from Figure 4.11 [1]

In HL7 V2 the semantics of a data item (field name) is defined implicitly by the message type, the segment and the position of a group or field within the segment. In order to create meaningful predicates in the RDF representation, a machine-readable format of the HL7 specification is required. Such a specification is provided by the HAPI project [HAP] which implements a generic parser for HL7 messages in the Java programming language. HAPI is utilized to parse each message and Java reflection is

[1]The schema definition and complete URIs have been omitted for better readability

used to traverse the resulting object model which provides field names. In this process, only these segments, groups and fields are transformed for which data exists within the message. Therefore, the resulting RDF representation is very compact as can be seen in Figure 4.13. In a real-world scenario, a large number of segments, groups and fields specified for the different message types are never used. Simply generating an RDF/S schema definition out of the HL7 specification would therefore result in a large volume of redundant metadata. In contrast, an incremental approach is implemented which always updates a global RDF/S schema description after processing a message.

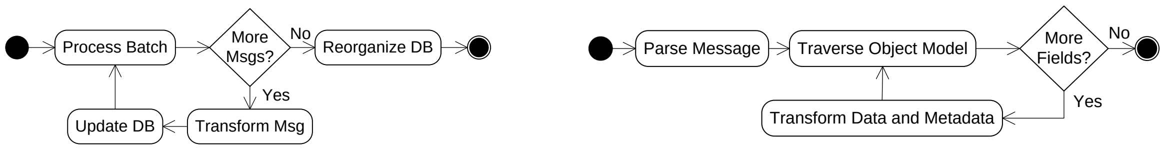


Figure 4.14: Processing (1) a batch of messages and (2) an individual message

Detailed UML activity diagrams depicting the described approach are shown in Figure 4.14. Messages are transformed in batches. To this end, the transformation can either be executed periodically or when a pre-defined number of messages has been received. When a batch is processed, each message is transformed into an RDF representation and the resulting data is passed to the underlying RDF database. Afterwards, the database is reorganized. This step is specific to the database system utilized by our implementation and will be explained in more detail in the following section.

When processing an individual message, it is first parsed into a Java object. The resulting object model is then traversed utilizing Java reflection. This process is implemented in a highly generic manner. The object model is traversed recursively and for each method it is checked whether it returns a valid result (e.g., not `null`). If it does return a valid primitive value, it is materialized in the resulting RDF graph by deriving a meaningful predicate identifier from the method name. Otherwise the returned object is traversed. In order to only retrieve data which is part of the actual HL7 message, we maintain a blacklist of methods which are excluded from this process (e.g., `equals(...)` or `toString(...)`). Identifiers for objects (see Figure 4.13) are generated incrementally, with identifiers for segments, groups and fields being defined relative to the current message identifier.

4.4.2 Evaluation

In this section we evaluate the performance of our solution with realisitic data characteristics. Because the data is extracted from a continuous message stream, the bottleneck in this process is the insertion of new data into the underlying database. The overhead induced by the transformation itself is negligible in this context. The experiments were performed on the same system as the experiments in the previous section.

As an RDF database system we used the RDF-3X triple store [NW10] because it is one of the most efficient open-source RDF database systems available and offers excellent performance. The transformation component is implemented in Java and executed on a 64-bit Sun JVM in version 1.6.0 with default settings. The transformation component and the RDF-3X database system communicated via standard input and output streams. As RDF-3X implements a highly compressed indexing of all possible permutations of the triples' subjects, predicates and objects (as well as all subsets) updating data in RDF-3X is not straight-forward. The system therefore implements a two-step process. When data is written to the database (*Update*) the system creates a so-called differential index. This means that the data is not directly inserted into the B^+-trees maintained by the system, but added to newly created additional indexes. As these additional indexes have a negative impact on the size of the index structures and also decrease the system's querying performance, RDF-3X implements a reorganization step in which the data from the differential indexes is merged with the main indexes (*Reorg*).

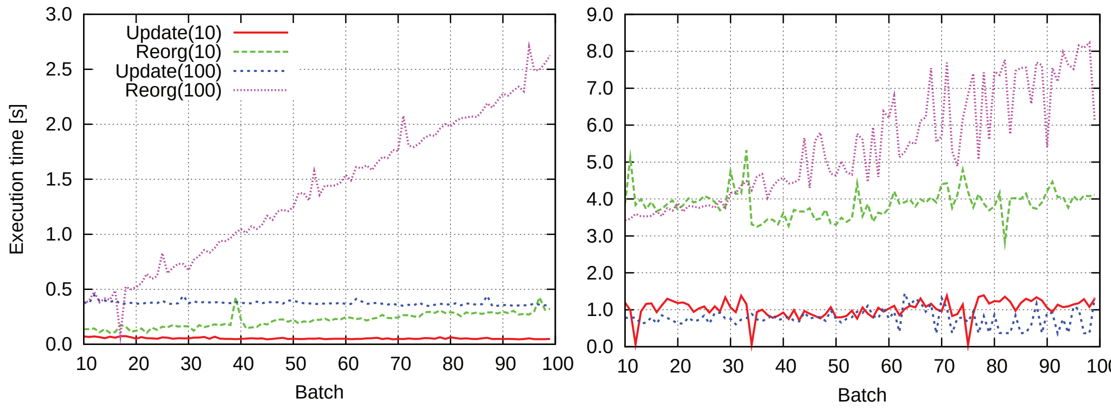


Figure 4.15: Execution times with (1) an empty dataset, (2) 200k triples

Our component receives messages via the FTP protocol. This means that it monitors a folder in the file system, which contains all received HL7 messages as plain text files. The experiments consisted of randomly created batches of HL7 ADT messages which were sent to the transformation component.

The benchmarks were performed on an empty database, as well as on a database with an initial size of 200.000 triples. Each experiment contained 100 batches with batch sizes of 10 and 100 messages. On average, each message resulted in about 30 triples. An experiment with the larger batch size corresponds to the volume of ADT data which is created within a large maximum care hospital in one week. As can be seen in Figure 4.15, the execution time of the update process mainly depends on the batch size and grows only slightly with the size of the database. In contrast, the execution time of the reorganization step clearly increases with the overall data volume. The solution scales very well and is easily able to handle realistic data volumes.

4.5 RDF Databases

Because RDF is the underlying data model of the developed prototype implementation, RDF databases can be integrated into the system without any additional efforts such as data transformation. This enables the inclusion of a large number of publicly available biomedical knowledge bases and vocabularies into the dataspace. In this section, we briefly review a few domain-specific datasets, which are important to the presented use cases. This includes datasets for drug developers and researchers, which, e.g., include information about proteins, diseases, genes, metabolic pathways, drugs and clinical trials.

System	Subject	Acronym
Infobox Properties[6]	Various	IP
Other Properties[6]	Various	OP
GeneID[7]	Genes	GID
Linked CT[5]	Clinical trials	LCT
HGNC [BNT$^+$08]	Unique gene symbols and names	HGNC
OMIM[1]	Disease genes and phenotypes	OMIM
Drugbank[2]	Chemical, pharmacological and pharmaceutical data	DB
Dailymed[3]	Marketed drugs	DM
Sider[4]	Adverse effects	SI
Diseasome [GCV$^+$07]	Disorders and disease genes	DI

Table 4.2: Example datasets from the Linked Open Data cloud

Relevant datasets from the Linked Open Data (LOD) are shown in Table 4.2. The LOD is a community effort to publish datasets according to the Linked Data principles. Additionally, significant efforts have been put into linking the datasets to each other. For example, each of the example datasets contains links to at least three other datasets, as can be seen in Table 4.3.

	IP	OP	GID	LCT	HGNC	OMIM	DB	DM	SI	DI
IP	-			X			X	X	X	X
OP		-		X			X	X	X	X
GID			-		X	X				X
LCT	X	X		-			X	X		X
HGNC			X		-	X	X			X
OMIM			X		X	-				X
DB	X	X		X	X		-	X	X	X
DM	X	X		X			X	-	X	X
SI	X	X					X	X	-	X
DI	X	X	X	X	X	X	X	X	X	-

Table 4.3: Links between the example datasets

Diseasome [GCV$^+$07] publishes a network of disorders and disease genes that have been obtained from Online Mendelian Inheritance in Man (OMIM). OMIM is a collection of human disease genes and their mutations. It also covers phenotypical information and links to scientific publications. Detailed chemical, pharmacological and pharmaceutical data is provided by DrugBank. Dailymed contains information about marketed drugs. Adverse effects are covered by SIDER, which contains information on marketed medicines and associated adverse reactions. LinkedCT provides data about clinical trials that have been obtained from a public clinical trial registry. DBpedia consists of information derived from Wikipedia which contains a rich collection of biomedical data. The Bio2RDF [BNT$^+$08] project publishes different datasets, containing a derivative of Entrez Gene (GeneID) which is a database for gene-specific information and the Human Genome Nomenclature (HGNC). The content of Entrez Gene contains data from different NCBI databases and collaborating model organism databases. HGNC defines a unique and meaningful name for every known human gene and assigns an abbreviation.

[1]http://www.ncbi.nlm.nih.gov/omim
[2]http://www.drugbank.ca/
[3]http://dailymed.nlm.nih.gov/
[4]http://sideeffects.embl.de/
[5]http://linkedct.org/about/
[6]http://dbpedia.org
[7]http://www.ncbi.nlm.nih.gov/gene (Relevant subset of about 20M interlinked triples)

4.6 Summary and Perspectives

This chapter presented comprehensive techniques for accessing relevant data from within RDF-based applications. This included a novel approach for accessing relational database systems, which are the predominant storage solution for clinical data, research data and other data such as knowledge bases. Compared to previous work, our approach offers a consistent way of mapping relational data to the RDF model, by utilizing context-specific operators. This eliminates the need to define mappings as a mixture of relational and Semantic Web concepts and facilitates the implementation of easy-to-use graphical editors. Because any mapping can be compiled into a set of independent SQL queries, the creation of RDF dumps is highly scalable as the system leverages the mature querying capabilities of modern database systems. Moreover, the approach allows data integrators to solely operate within the RDF world.

Future research in this context could investigate how the proposed techniques are suited for directly accessing the underlying database by means of query rewriting. In theory, this could be implemented very efficiently as any SPARQL query could be translated to a set of independent SQL queries whose results simply need to be merged. A remaining challenge is how to avoid duplicate results and return correct cardinalities for the resulting variable bindings. This either requires a fine-grained user-defined specification of which metadata should be included in which fragment or an automatic technique to restrict the export of metadata for each RDF class to a certain fragment by analyzing the extraction rules as well as the dataset. Our current implementation is not able to automatically detect dependencies between the data if a databases' metadata catalog is incomplete and does, e.g., not specify foreign key constraints. Future work could investigate techniques to detect such dependencies by analyzing the database [BLNT07].

In addition to relational databases, clinical and administrative data from the context of patient care can also be accessed via HL7 message streams. The presented approach allows to automatically and scalably materialize a heterogeneous stream of HL7 messages into an RDF dataset. By utilizing a generic process, which traverses object models generated by an HL7 parser, the approach is able to transparently integrate HL7 messages with varying schemas and even different versions of the HL7 standard. We have also shown that modern RDF databases are able to handle HL7 message streams with realistic data volumes. The component is able to support the cleansing of identifying data, which is an essential requirement for biomedical integration systems. Furthermore, it implements a lightweight approach for accessing clinical data in RDF format, as it allows to automatically transform clinical message streams. Therefore, the component provides access to important data for translational research.

A promising direction for future research would be to investigate, how the ontological representation of HL7 V2 messages can be utilized to (at least partially) harmonize their RDF representation with the HL7 V3 RIM and related ontologies. Such a harmonization would allow to utilize the rich semantics of HL7 V3 for data integration and potentially provide means to foster interoperability.

An increasing number of RDF representations of publicly available datasets are created and linked to each other. These sources of information can be integrated into the data integration platform without any additional efforts and we have presented a brief overview over important public RDF-datasets covering our application domain.

CHAPTER 5

Maintaining Local Autonomies: Distributed Query Processing

The integrated access to different RDF datasets is one of the key components of the system presented in this thesis. It is not only the primary interface available to applications or end users but also forms the basis of additional concepts for semantic integration. Our approach implements loose coupling by spreading different data and metadata over different RDF repositories. This separation can be utilized to implement a fine grained permission model while the ability of RDF to reference arbitrary data items within other datasets is utilized to „glue" the data together. It is therefore very important to provide efficient means for executing queries within this environment. This chapter focusses on answering conjunctive queries over a set of distributed RDF databases, as these form the backbone of any SPARQL query processor.

According to Chapter 3, $T = I \cup B \cup L$ denotes the set of all RDF nodes (i.e., URIs (I), blank nodes (B) and literals (L)) and V is the set of all variables. Inspired by [PAG09], we first define a function which executes a conjunctive SPARQL query $Q = \{t_0, ..., t_n \mid t_i \in T \cup V \times I \cup V \times T \cup V\}$ over an RDF graph G. A mapping m is a partial function $V \to T$. For a triple pattern $t \in Q$, $m(t)$ is the triple $g \in G$ obtained by replacing all variables in t according to m. The domain of m, $dom(t)$, is the subset of V for which m is defined. Two mappings m_1 and m_2 are compatible if for all $x \in dom(m_1) \cap dom(m_2)$ the following holds true: $m_1(x) = m_2(x)$, i.e., $m_1 \cup m_2$ is also a mapping. Let M_1 and M_2 be sets of mappings. The join $M_1 \bowtie M_2$ is defined as $\{m_1 \cup m_2 \mid m_1 \in M_1$ and $m_2 \in M_2$ are compatible mappings$\}$. Furthermore, $var(t)$ returns the set of all variables contained in a triple pattern t. The function $S(G,\ Q)$ which executes Q over G is defined recursively:

1. $S(G,\ \{t\}) = \{m \mid dom(m) = var(t)$ and $m(t) \in G\}$
2. $S(G,\ \{t_0, t_1, ..., t_n\}) = S(G,\ \{t_0\}) \bowtie S(G,\ \{t_1, ..., t_n\})$

Within a distributed environment, the queried RDF graph G consists of several datasets, i.e., $G = G_0 \cup G_1 \cup ... \cup G_n$. This section will describe a novel RDF-specific synposis as well as efficient compile-time and and run-time optimizations for the distributed execution of SPARQL queries within our environment. Our concept aims at integrating those potentially diverse datasets without requiring upfront schema-level knowledge. It is therefore based upon a purely syntactical summary which can be generated automatically. The remainder of this chapter is structured as follows. Section 5.1 discusses related work whereas Section 5.2 presents our synopsis. Sections 5.3 and 5.4 cover query optimization and execution. The chapter is concluded by an evaluation in Section 5.5 and a discussion of the results in Section 5.6.

5.1 Related Work

Only recently, systems have become available that allow for scalable and efficient management of large RDF graphs. Most notably the RDF-3X system [NW10] achieves very good performance by utilizing exhaustive indexing of all possible permutations of RDF triples and fast merge-joins combined with RDF-specific techniques for join-processing and query optimization [NW09]. Although systems like RDF-3X are well suited for querying local RDF data sources, the Semantic Web is distributed by nature. Despite this inherent distribution, systems for querying distributed RDF data sources are still in their infancy. Some highly specialized solutions (e.g., [BB10, LH10, LT10]) are not applicable in our context. This section provides a comprehensive overview over generic approaches for querying distributed RDF repositories. These can be categorized along different axes, which are shown in Figure 5.1.

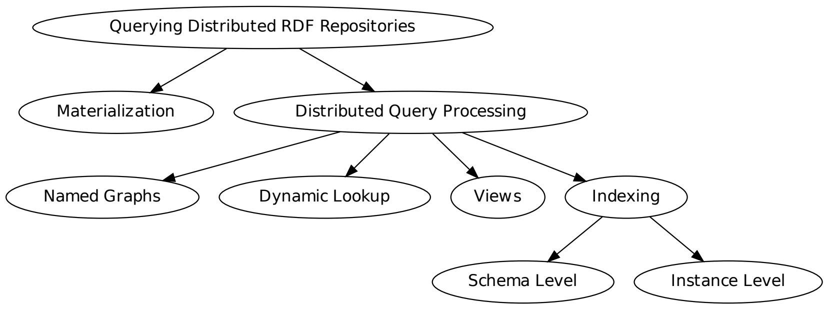


Figure 5.1: Techniques for querying distributed RDF repositories

Materialization

Materialization-based approaches such as semantic web search engines offer an integrated view on distributed RDF data sources by crawling the Semantic Web into a central physical repository, e.g., [HHUD07, ODC$^+$08]. As local query processing is generally much cheaper than distributed query processing these systems can offer excellent response times. Typical drawbacks of this approach include outdated information as collecting and indexing is a time consuming task, and the loss of the data sources' access autonomies. Especially in the biomedical domain the loss of sovereignty is critical, as restrictions have often to be enforced due to legal and regulatory requirements or issues of intellectual property rights.

In contrast to materialization-based systems, approaches for *distributed query processing* offer means to preserve these autonomies as data is still located at its origin and global queries are evaluated by accessing the remote data sources on demand. The key idea is to derive local subqueries from the original global query. These subqueries are then answered by the local data sources and the global result is computed by processing the local results. In this context, query optimization techniques aim at deriving exactly those subqueries, which are needed to actually answer the query and try to find an efficient evaluation strategy.

Named Graphs

A very simple form of distributed query processing can be implemented by utilizing the RDF notion of *named graphs*. This has been implemented in [CFM$^+$09] for the life sciences domain. RDF allows to assign identifiers to datasets (i.e., a named graph), which can be utilized to explicitly specify which parts of the query are to be evaluated at which endpoint. Despite the fact that this approach is easy to implement, it has numerous drawbacks. It does not offer location transparency and queries can only be defined by domain experts with in-depth knowledge of the available data sources. Moreover, the formulation of a query is very error-prone and time-consuming. Due to the lack of information about the actual content of the data sources, it is very difficult to implement any query optimization techniques.

Dynamic Approaches

This category describes systems which do not require any additional information, such as graph-identifiers, views or synopses, but answer a query dynamically without any information about the data sources. The dynamic-lookup approach has been proposed in [HBF09]. It implements a query execution concept, which has been explicitly designed for data sources that implement the *Linked Data* principles. The *direct lookup* approach traverses this network of data elements during query execution and answers the query based upon the collected information. It starts by dereferencing URIs which are included in the original query itself. As heuristics are utilized to reduce the number of expensive HTTP requests, such systems often return incomplete results which is not acceptable in our context.

FedX is a federation layer for distributed RDF databases, that is oriented towards a volatile Linked Data scenario [SHH$^+$11]. Query optimization is performed at runtime and driven by heuristics. Data localization is implemented with special SPARQL queries (`ASK`). In advance to query execution, each endpoint is asked for which triple patterns it can return a result. Similar to other approaches, the volume of intermediate results is reduced by applying the bind join mechanism. The basic idea is to bind a variable in a subquery to all relevant values obtained by executing the previous subqueries to increase its selectivity. It is the most important query execution technique in all state-of-the-art distributed SPARQL query processors. Although its data localization technique is not very accurate, *FedX* achieves excellent performance by employing a multithreaded approach that executes multiple subqueries in a highly parallel manner. As the system does not require preprocessing, it is well suited for dynamic environments.

SPLENDID implements a hybrid approach [GS11]. It also utilizes SPARQL `AKS`-queries for data localization. In addition, it maintains *VoID* (Vocabulary of Interlinked Datasets) descriptions of the integrated RDF databases [VOI]. VoID is an RDF vocabulary that has been developed to provide meta-level descriptions of RDF datasets, for discovering relevant datasets and not for distributed query processing. The proposed approach utilizes this information for join ordering and more accurate data localization. The most efficient variant of the system, which extensively utilizes VOID data, is not able to handle schema-free datasets, though. For query processing, the system also relies on the bind join operator.

Views

Systems which are based on *views* implement a concept similar to federated relational database systems [Kos00]. The schemas of the local data sources are aligned by either mapping them to a global schema (local as view), or by building a global schema out of the local schemas (global as view) [Hal01]. This can, e.g., be implemented via declarative mapping rules, or by extending the SPARQL query language. Query processing can either be implemented by query rewriting (e.g., [MBG+09]) or by inference and materialization [SS08]. Some approaches implement peer data management systems (PDMS), which integrate distributed databases by mapping the local schemata to each other [XC04]. This allows to implement a cascading query execution process. Although these systems do not only offer distributed query processing but also means to semantically integrate the datasets, they require an explicit schema definition of the datasets. This contradicts the schema-free nature of RDF and the incremental integration process advocated in this work.

Indexing

Indexing-based approaches utilize synopses or statistical information about the data. Different index structures can further be be categorized by whether they only include schema-level information (i.e., RDF predicates) or also cover instance-level information.

Schema-level Indexing: In [SVHB04] the authors propose a lightweight approach, which implements data localization (i.e., deciding which parts of the query can be answered by which data source) based on information about *RDF predicates* contained in the datasets. The proposed structure is oriented towards object-oriented database systems. It indexes all schema-paths (i.e., chains of attributes) and organizes them in a hierarchy of paths and subpaths. The drawbacks of the approach include that predicates always have to be bound, too many sources are selected for common predicates and query optimization is difficult due to the lack of instance level information. In addition, the system is limited to answering path-structured queries.

Instance-level Indexing: Instance-level summaries do not only focus on RDF predicates but also cover subjects and objects. *DARQ* implements a schema-level synopsis which is combined with instance-level data, i.e., information about the objects associated with a predicate [QL08]. This includes, e.g., expressions that can be defined by SPARQL filter operators (e.g., only names which start with the letters A-F). Furthermore, cardinalities (i.e., the number of subjects which appear in a triple containing the predicate) and selectivities (i.e., the number of subjects for different object values) are stored. The system is able to extract the required instance-level information automatically and also offers interfaces for manual fine tuning by experts. Although this additional information enables the query processor to perform some optimizations, the synopsis lacks comprehensive instance-level information. As the approach also utilizes predicates for source selection, it is restricted to answering queries with bound predicates.

SemWIQ utilizes more detailed statistical information about the data sources [LWB08, LW09a]. The system maintains histograms over all values of predicates for all contained RDFS classes [LW09a]. It distinghuishes between histograms for different XML Schema data types. If a predicate's values are RDF resources, the URIs are not compressed or approximated. The synopsis is generated automatically and allows to estimate the selectivity and cardinality of individual triple patterns and filter expressions. These estimates are utilized for join-order optimization. The query execution engine further implements several well-known optimization techniques, such as semi-joins and row-blocking [Kos00]. The approach imposes several restrictions on datasets and queries which make it unfeasible for a deployment in our context. For example, schema information has to exist for each data source and instances have to be annotated with it. Moreover, the approach is limited to SPARQL queries with variable subjects and explicitly defined RDFS classes. This is necessary to associate the variables in the query to the histograms in the source index. Because URIs are not compressed, the approach can not handle large datasets.

Data Summaries

Because of the wide-spread use of some vocabularies and the schema-relaxed nature of SPARQL, many RDF databases are potentially able to answer a single triple pattern. But if the same triple pattern is part of a more complex SPARQL query many of these answers are irrelevant due to a lack of join partners. Because of the design of their instance-level synopses, the previously described systems are not able to prune the resulting redundant subqueries. A system which is able to consider such dependencies and to answer arbitrary queries (e.g., with unbound predicates) has been presented in [HHK+10]. Here, data localization is implemented based on a combined schema- and instance-level synopsis. The summary is built by transforming the RDF data into a numeric space by hashing subject, predicate and object independently. The resulting points are then approximated by a spatial index structure consisting of three-dimensional minimum bounding boxes (MBBs). The index structure implements a Q-Tree [HKK+07], which is a variant of an R-Tree [Gut84]. Such trees do not reference the indexed data but approximate it. Q-Trees have also been described as a mixture of R-Trees and histograms [HHK+10]. When optimizing a given SPARQL query, constants contained in a triple pattern are hashed and a point- or range-query is performed on the synopsis, returning a set of minimum bounding boxes (MBBs). Dependencies between triple patterns are taken into account by executing the same operations (i.e., joins) on the resulting MBBs, as would have been performed on the variable bindings during query execution. Therefore sources can be pruned that returned MBBs which did not have a join partner in any of these operations.

In the following sections a novel synopsis is presented, which is inspired by the concept of RDF data summaries but allows to optimize much more complex queries. One important aspect of our synopsis is that it includes type information. This allows the optimizer to produce more accurate results. Furthermore, an adequate conversion of literals into numeric values allows the consideration of many SPARQL operators into the query optimization process. Approximating several triples by ranges of hash-values is inaccurate, as this often leads to overlapping MBBs, that do not approximate the same RDF nodes. Our novel data structure contains more detailed information but still remains very compact. Finally, we present a distributed SPARQL query processor that implements a comprehensive approach by employing different query execution mechanisms that have previously been studied in an isolated manner. To this end, the optimizer utilizes the summarization, to obtain in-depth knowledge about the characteristics of a query. This information is then utilized to further adopt an optimized execution plan to a given query. This includes different techniques for distributed join processing and novel query optimizations that allow to reduce the number of subqueries and to increase their selectivity.

5.2 Indexing

The basic idea of an RDF Data Summary is to project the data onto a set of three-dimensional points by computing independent hash values for each triple's subject, predicate and object. These points are then approximated by a spatial index structure called Q-Tree [HHK+10]. Q-Trees are a variant of R-Trees that have a constant number of leaf nodes and do not reference the data items themselves but summarize them, e.g., by storing counts. Data items are approximated by *Minimum Bounding Boxes* (MBBs) on leaf level. A set of nodes is itself summarized by a MBB on a higher level, up to one single root node. Figure 5.2 shows an example Q-Tree.

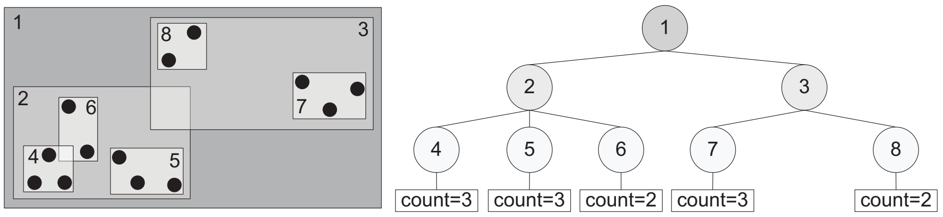


Figure 5.2: A two-dimensional Q-Tree

A data summary is a three-dimensional Q-Tree for all data sources, which is built by iteratively transforming the triples from each source and inserting them into the tree. Each leaf node stores a set of source identifiers, including one for each source of a triple approximated by the node. During query optimization a range-query is built from each triple pattern's constants and variables and executed against the synopsis. The result of such a query is a set of MBBs derived from valid leaf nodes. If the result is not empty, every system that is referenced by any of the resulting MBBs could potentially return variable bindings for the according triple pattern.

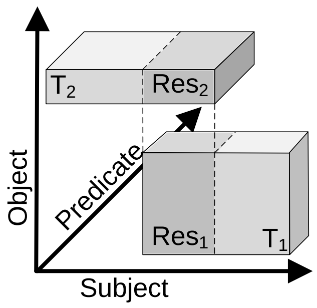


Figure 5.3: Join of two MBBs over the subject-dimension

Although this approach is highly flexible, it is not well suited for complex queries and large datasets. As the nodes of the indexed RDF datasets are hashed, any type information is lost. However, the preservation and incorporation of type information would lead to more accurate results (e.g., resources can only be joined with resources) and the ability to consider further SPARQL operators (e.g., selecting results with filter expressions). In addition, it is very inaccurate to approximate several triples by ranges of hash values, as this often leads to overlapping MBBs that do not approximate common RDF nodes. This is reinforced by the fact that Q-Trees are designed to have a predefined, constant number of leaf nodes regardless of the size of the dataset. In general there are therefore considerably more join partners for MBBs than there would be for the approximated triples. This leads to inaccurate results and limits the optimizer to simple queries and small datasets due to significant main memory requirements and running times.

5.2.1 Type Information

Our data structure incorporates information about the type of resources and literals and implements value- or order-preserving transformations for some literal data types. This leads to more accurate results and allows to consider additional SPARQL operators during query optimization. Type information is encoded as two-byte unsigned integers representing type-identifiers. Therefore 65536 different data types can be distinguished. For common literal data types (such as integers, strings or calendar dates) type-identifiers and transformations are assigned statically. Strings are further distinguished by their defined language. Values with data types which can be encoded by the eight bytes provided for hash values, such as integers, floats, calendar dates and timestamps, are kept as is and interpreted accordingly.

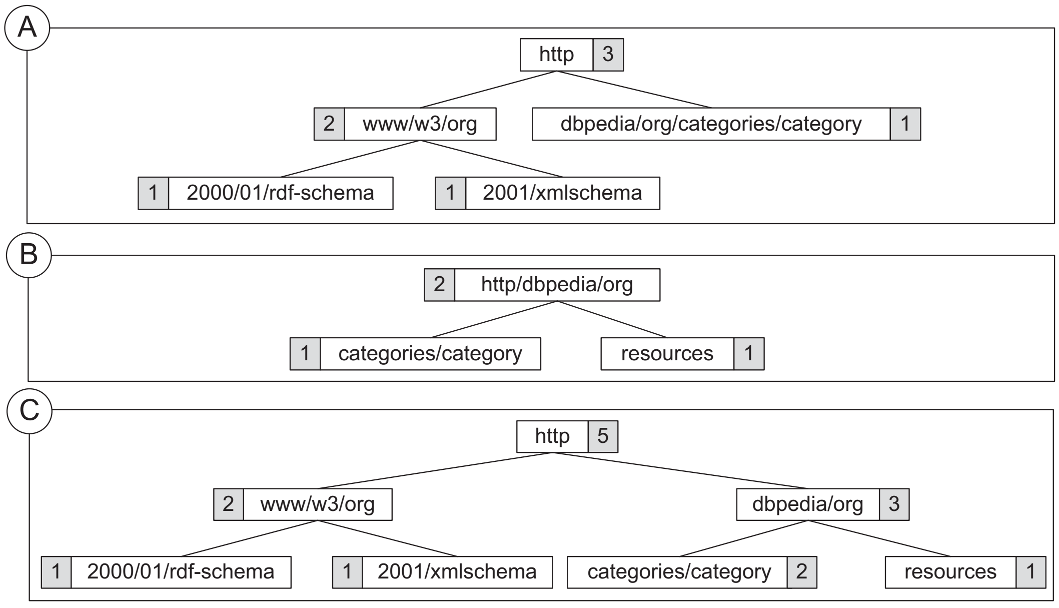


Figure 5.4: Local and global prefix trees

Type-identifiers are not only used for literals but also for resources. Type information can be assigned to resources by encoding a prefix of the resources' URIs. In contrary to literals, this can not be implemented by simply assigning identifiers to these prefixes as there might be too many of them (e.g. a very heterogeneous collection of URLs). However, as even datasets that do not adhere to a schema very often preferably use a certain vocabulary, most URIs in an RDF dataset share a set of *common prefixes*. To determine the common prefixes of an individual dataset we normalize the triples' URIs and split them into a list of path components:

- `http://dbpedia/org/categories/Category:Antidiabetic_drugs`
- `http-dbpedia-org-categories-category-antidiabetic_drugs`

We then add all components (excluding the last element) to a radix tree and count the number of their occurrences. Finally, a global view is needed as this allows to map the same prefixes from different datasets onto the same type identifiers. Local radix trees are computed in parallel for each data source (trees (a) and (b) in Figure 5.4) and merged into a global tree (see tree (c) in Figure 5.4). Each leaf node represents one prefix which can be built by appending all components on its path to the root node. If the number of prefixes is larger than what can be encoded by the available two bytes, the top-k prefixes can be selected by iteratively removing the leaf node with the lowest frequency until only k leaf nodes remain.

The organization of common prefixes in a radix tree also allows to efficiently determine the type for a given normalized URI as radix trees support an efficient longest prefix matching operation. Parts of the URI not contained in the longest prefix are transformed by applying a hash function. To cover unknown or infrequent types we reserve default identifiers. In the following sections we assume that for any resource, blank node or literal $n \in T$:

- $\mathit{type}(n)$ returns the two-byte type identifier, and
- $\mathit{hash}(n)$ returns the eight-byte hash value.

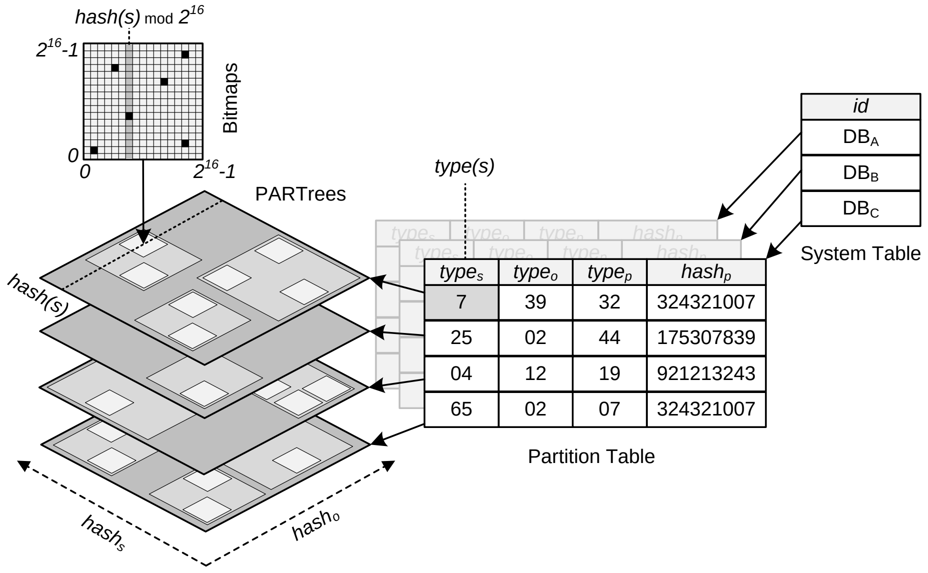
5.2.2 Data Organization

The proposed data structure does not implement a single synopsis for all datasets, but one summary per data source. This allows for independent updates and increases the accuracy as any MBB can be uniquely associated with a single endpoint. Furthermore, we apply vertical partitioning [AMMH07] to each summary. This concept is based on the observation that for real-world biomedical RDF datasets the number of distinct predicates is very small compared to the number of triples. We therefore group triples that share the same predicate into a common partition, efficiently eliminating the need to store most of the predicates. This enables us to reduce the amount of redundant information and at the same time preserve the predicates' hash values. This further increases the accuracy on schema-level, which is important because predicates are rarely unbound in common SPARQL queries [AFMPdlF11,PV11]. Moreover, vertical partitioning allows to efficiently handle type information. For this purpose, we partition the dataset not only by predicate hash value, but also by subject, predicate and object type. As a result, a partition $P = (\mathit{type}_s, \mathit{type}_p, \mathit{type}_o, \mathit{hash}_p)$ references a two-dimensional spatial index structure, approximating subject and object hash values for any triple $t = (t_s, t_o, t_p)$ with $\mathit{type}(t_s) = \mathit{type}_s$, $\mathit{type}(t_p) = \mathit{type}_p$, $\mathit{hash}(t_p) = \mathit{hash}_p$ and $\mathit{type}(t_o) = \mathit{type}_o$. An example for our synopsis is show in Figure 5.5. The system table references a list of partitions (partition table) for each indexed data source. Each of these partitions again references a spatial index structure that will be explained in more detail in the following section.

5.2.3 Partition Trees

In order to further increase the accuracy of our summary we developed an extension of Q-Trees, which we call Partition-Trees or PARTrees for short, that store more comprehensive information about the data elements in one partition. Conceptually, sets of integers are added to the leaf nodes which store the two least significant bytes (LSBs) of the approximated hash values. For example, for the subject-dimension the set b_s contains the value $(\mathit{hash}(t_s) \mod 2^{16})$ for each triple $t = (t_s, t_p, t_o)$ approximated by the leaf node. Resulting in only a tiny space overhead, it is further possible to preserve the correlation between the LSBs of the hash values for the different dimensions. For this purpose we store a set of points $n.\mathit{lsb} = \{(\mathit{hash}(t_s) \mod 2^{16}, \mathit{hash}(t_o) \mod 2^{16})\}$

that approximate the subject and object hash value of each triple $t = (t_s, t_p, t_o)$ summarized by a leaf node n. Because each of these points occupies two bytes for the subject and object dimension respectively, the structure consumes at least four bytes of memory per triple. As can be seen in the following sections this growth in memory consumption is legitimate because the stored information helps to determine sources for triple patterns more accurately and drastically reduces the number of join partners for most MBBs. In contrast to Q-Trees, which have a fixed number of leaf nodes, we allow our trees to grow with the number of indexed data elements. In order to control space consumption we define a maximum fanout for inner nodes and leaf nodes which describes the maximum number of child nodes (or approximated triples) per node. During indexing the PARTrees are generated locally by applying the Sort-Tile-Recursive (STR) bulk-loading algorithm [LEL97] and transmitted to the mediator on a per-partition basis where they are kept in main memory.



id
DB_A
DB_B
DB_C

$type_s$	$type_o$	$type_p$	$hash_p$
7	39	32	324321007
25	02	44	175307839
04	12	19	921213243
65	02	07	324321007

Figure 5.5: Global synopsis

5.3 Query Optimization

The initial execution plan is generated by selecting possible sources for each triple pattern. To this end type information and hash values are derived from the constants contained in the patterns. The global synopsis is now queried in order to determine potential sources. A partition $p = (type_s, type_p, type_o, hash_p)$ is valid for a triple pattern $t = (t_s, t_p, t_o)$ if its referenced PARTree might approximate a triple that satisfies t. This means that the types and hash value defined by the partition p match the triple pattern's constants, i.e., all of the following conditions hold true:

1. t_s is a variable, or $type(t_s) = p.type_s$,
2. t_p is a variable, or $(type(t_p) = p.type_p$ and $hash(t_p) = p.hash_p)$,
3. t_o is a variable, or $type(t_o) = p.type_o$.

For each system the partition table is checked and range- or point-queries are executed against the PARTrees referenced by valid partitions. Similar to R-Trees, the trees are traversed from the root node to the leaf nodes on each path consisting of valid nodes

n with $n.min_d \leq hash(t_d) \leq n.max_d$ for each constant dimension $d \in \{s, o\}$. A leaf node $n_l = (min_s, max_s, min_o, max_o, lsb)$ is valid if its MBB also covers $hash(t_d)$ for each constant dimension $d \in \{s, o\}$ and the stored LSBs match the hash value(s), i.e., the following holds true:

1. $t.s$ and $t.o$ are variables, or
2. $t.o$ is variable and $(hash(t.s) \mod 2^{16}, y) \in n_l.lsb$, or
3. $t.s$ is variable and $(x, hash(t.o) \mod 2^{16}) \in n_l.lsb$, or
4. $(hash(t.s) \mod 2^{16}, hash(t.o) \mod 2^{16}) \in n_l.lsb$.

Based on these preconditions it is possible to determine the set of potential sources for each triple pattern from the global synopsis which is sketched in Algorithm 1. If the optimizer is not able to determine at least one potential source for each of the query's triple patterns, the query can not yield any results and additional optimization and query execution can be omitted.

This process is exemplified for a constant subject-dimension in Figure 5.5. Here, we assume that we query the data summary in order to find sources for the triple pattern *(<http://dbpedia.org/resources/BarackObama> ?p ?o)*. First, we transform the constant subject by matching a normalized representation against the prefix tree. For *s=http://dbpedia.org/resources/BarackObama* we obtain *type(s) = type(http-dbpedia-org-resources-BarackObama) = type(http-dbpedia-org-resources/)*, which we assume to be 7. The according hash value is retrieved by applying the hash function to the remainder of the URI, i.e., *hash(s) = hash(BarackObama)*. As is shown in Figure 5.5, *type(s)* is first utilized to determine valid partitions for each registered data source. If a valid partition is found, the associated PARTree is traversed by means of *hash(s)* down to valid leaf nodes. This process is repeated for all datasets and partitions referenced in the synopsis.

Algorithm 1: Source Selection

```
Input: Triple pattern $t$
Result: All possible sources for $t$
begin
    $S \leftarrow \emptyset$
    foreach system $s$ do
        foreach partition $p$ of $s$ that is valid for $t$ do
            if tree of $p$ contains valid leaf node for $t$ then
                $S \leftarrow S \cup \{s\}$
                break
    return $S$
```

For our example query from Figure 3.5, sources for the triple pattern `(?compound, formula, ?formula)` can be found by only checking the partition tables as both subject and object are variables. For the triple pattern `(?compound, contraindication, "Hypovolemia")` additional range queries with a constant at the object dimension have to be performed.

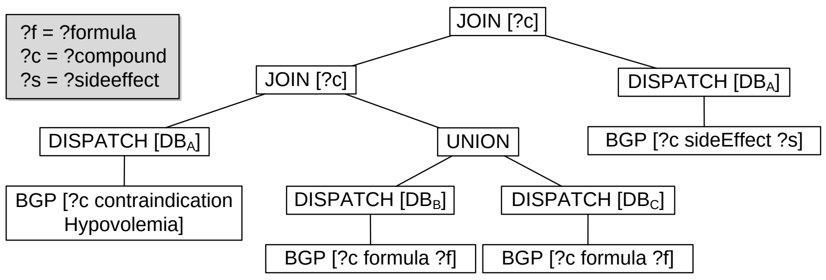


Figure 5.6: Initial plan for the example query

An initial execution plan is built from the set of potential sources by modeling dependencies between patterns as joins over its variables. The operator $Join(V_j)$ denotes that the variable bindings returned by its children are to be joined over variables V_j. An operator $Dispatch(s)$ states that its subtree is to be evaluated by the remote system s. The case in which there is more than one source for solutions to a triple pattern is modeled by unifying the results of different dispatch operators. The operator $Union$ therefore denotes that the variable bindings returned by its children are to be unified. The initial execution plan for our example query is shown in Figure 5.6. We assume a scenario consisting of three data sources, DB_A, DB_B and DB_C in which the triple pattern `(?compound, formula, ?formula)` can be answered by the systems DB_B and DB_C whereas the other two triple patterns can only be answered by the system DB_A.

5.3.1 Plan Simplification

In the next optimization phase the initial execution plan is simplified by *simulating* its execution based on the information provided by the data summary. The variable bindings returned by evaluating a triple pattern at a remote system are represented by *Approximate Variable Bindings* (AVBs). Each AVB b approximates a set of bindings for a set of variables V by defining type information, boundaries for hash values and compressed bitsets for each variable dimension $v \in V$.

1. $b.min_v$ is a lower bound for the bindings' hash values.
2. $b.max_v$ is an upper bound for the bindings' hash values.
3. $b.type_v$ defines the bindings' type.
4. $b.bitset_v$ encodes the bindings' LSBs and must contain $(hash(n) \mod 2^{16})$ for a valid binding n.

The set of possible LSBs for the approximated variable bindings are encoded in a bitset (of length 2^{16}) because this allows to efficiently join two AVBs. As the bitsets are only sparsely populated, they can also be compressed very well. Each AVB is initially being produced by a certain dispatch operator as explained below. When an AVB has been completely processed and reaches the top of the query execution plan there are in general multiple dispatch operators that have contributed to it. Each AVB stores a bitset $b.dispatches$ that is used to keep track of these contributing operators. In the following sections we will describe these operators in more detail. As a $Union$ operator simply unifies the two sets of AVBs produced by its children, we focus on the $Dispatch$ and $Join$ operator.

Dispatch Operator

When executed, the triple pattern referenced by a dispatch operator *Dispatch*(s) is evaluated in the same way as during source selection but for a single source s. Similar to the initial plan generation process, the partition table of s is checked and the referenced PARTrees are queried for valid leaf nodes. The resulting leaf nodes are then converted into AVBs representing sets of potential bindings for a triple pattern. This process is sketched in Algorithm 2. For each variable dimension boundaries for hash values as well as the data type and a bitset is defined. If the predicate is a variable this information is derived from the partition's values for $type_p$ and $hash_p$. In case of a variable at the subject or object position the minimum and maximum boundaries for the bindings' hash values are defined by the leaf node's MBB and the type is defined by the partition.

Algorithm 2: PRODUCE AVBS

```
Input: System $s$, triple pattern $t = (t_s, t_p, t_o)$
Result: All AVBs for $t$ from system $s$
begin
    $B \leftarrow \emptyset$
    foreach partition $p$ of $s$ that is valid for $t$ do
        foreach leaf node $n$ in tree of $p$ that is valid for $t$ do
            $b \leftarrow new\ AVB$
            foreach $d \in \{s, p, o\}$ such that $t_d$ is variable do
                if $d = p$ then
                    $b.type_{t_d} = p.type_p$
                    $b.min_{t_d} = b.max_{t_d} = p.hash_p$
                    $b.bitset_{t_d}.set(p.hash_p \mod 2^{16})$
                else
                    $b.type_{t_d} = p.type_d$
                    $b.min_{t_d} = n.min_d$, $b.max_{t_d} = n.max_d$
                    $b.bitset_{t_d} = \text{DERIVEBITSET}(d, n, t)$
            $B \leftarrow B \cup \{b\}$
    return $B$
```

Bitsets for the subject and object dimension can be derived directly from the set of points *n.lsb* if both subject and object are variables. Otherwise the hash value of the constant dimension is used to project onto relevant bits for the variable dimension. If both, subject and object, are variables each dimension is transformed into a bitset by simply setting the bits as defined by the points' coordinates for the subject and object dimension. In case of one constant and one variable dimension only these bits are set for the variable dimension for which a point exists in *n.lsb* that has a coordinate matching the constant dimension. Algorithm 3 implements this for a variable subject dimension, the object dimension is handled analogously.

The downside of this transformation is that the correlation between the LSBs is lost if both subject and object are variables, resulting in possible false positives. On the other hand AVBs can now be joined much more efficiently, as the intersection of two bitsets can be performed simply by computing a bitwise AND-operation. Ascending integers are assigned as unique identifiers to each dispatch operator in order to enable the tracking of contributing sources. To this end the i-th bit is initially set in $b.dispatches$ for each AVB b produced by a dispatch operator with identifier i.

Algorithm 3: Derive Bitset

```
Input: Dimension $d \in \{s, o\}$, leaf node $n$, triple pattern $t = (t_s, t_p, t_o)$
Result: Bitset for the dimension $d$
begin
    $b \leftarrow new\ bitset$
    if $d = s$ then
        /* handle subject dimension*/
        if $t.o$ is a variable then
            foreach $(x, y) \in n.lsb$ do
                $b.set(x)$
        else
            foreach $(x, hash(t_o) \mod 2^{16}) \in n.lsb$ do
                $b.set(x)$
    else
        /* handle object dimension*/
    return $b$
```

Join Operator

Two MBBs can be joined if the defined ranges overlap for each join dimension. For AVBs we extend this by taking data types and bitsets into account. Two AVBs b_1 and b_2 can be joined if the following holds true for each join dimension $v \in V_j$:

- $b_1.min_v \leq b_2.max_v$ and $b_1.max_v \geq b_2.min_v$ (MBBs overlap),
- $b_1.type_v = b_2.type_v$ (types are equal),
- $(b_1.bitset_v$ AND $b_2.bitset_v) \neq 0$ (at least one equal bit is active).

The minimum and maximum boundaries for hash values resulting from a join between two AVBs are defined in the same way as for a spatial join of the MBBs involved. Moreover, the bitsets of the resulting AVB are defined as the result of a bitwise AND-operation on the bitsets of both AVBs for each dimension. Therefore the number of active bits in the AVBs' bitsets decreases as AVBs are passed upwards towards the root of the query execution plan. The properties of the AVB b_r resulting from a join between two AVBs b_1 and b_2 for each dimension $v \in V_j$ are defined as:

- $b_r.min_v = max(b_1.min_v, b_2.min_v)$,
- $b_r.max_v = min(b_1.max_v, b_2.max_v)$,
- $b_r.bitset_v = b_1.bitset_v$ AND $b_2.bitset_v$,
- $b_r.type_v = b_1.type_v$.

The bitset $b_r.dispatches$ is set to ($b_1.dispatches$ OR $b_2.dispatches$) in order to keep track of the sources contributing to b_r. Properties for variable dimensions not contained in V_j are simply inherited from either b_1 or b_2. Note that it is not possible that the underlying triple patterns contain a common variable that is not in V_j due to the implicit definition of joins. When joining two sets of MBBs we first partition each operand into sets of AVBs with equal data types in the join dimensions. We then solve the rectangle intersection problem for each combination of compatible partitions by applying a standard plane-sweeping algorithm. Afterwards the bitsets of each pair of intersecting AVBs are checked for compatibility.

Join-order Optimization

It is obvious that the performance of joining two sets of AVBs strongly depends on a good join-order for many queries. We therefore implemented a cost-based optimizer utilizing a top-down enumeration strategy with memoization [FM11]. We assume independence between the individual triple patterns and estimate selectivities based on the higher-level inner nodes of the PARTrees. For a given triple pattern we construct a set of MBBs by traversing the PARTrees of each valid partition up to a predefined depth (e.g., three). We estimate the selectivity for a join between two triple patterns by computing the cardinality of the spatial join between the sets of MBBs returned by this scans. As LSBs are only stored in the leaf nodes, we can in this process only consider the MBBs' and the data types defined by the partitions. This procedure has some potential drawbacks. Firstly, the PARTrees' higher-level nodes approximate a potentially large number of triples from which only a tiny fraction might be a valid solution. Therefore, the resulting selectivity estimation can be imprecise. Secondly, the assumption of independence of the selectivities of joins in a SPARQL query does not hold as has been shown in [NM11]. Despite these limitations, the method performs very well for most queries, as can be seen in Section 5.5.

5.3.2 Sideways Information Passing

Sideways information passing (SIP) techniques aim at reducing the volume of intermediate results and thus join complexities [NW09]. The general idea is to incorporate lightweight mechanisms, which allow for passing information sideways in the query execution plan, i.e., between operators that are neither ancestors nor descendants of each other. This can then be utilized to remove tuples from local intermediate results, which are known to not produce any results due to the data provided by other operators.

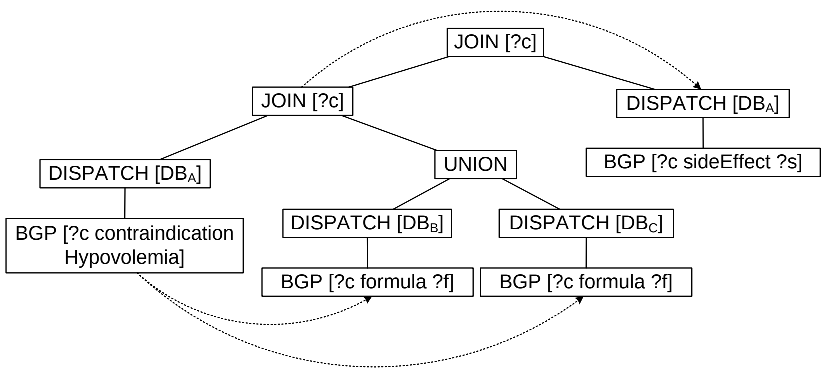


Figure 5.7: Sideways information passing

During query optimization, our system has to join potentially large volumes of AVBs. Unfortunately, this can not be implemented in a pipelined manner as two AVBs can only by joined over a certain dimension if their type identifiers are equal, the defined ranges overlap and at least one equal bit is set in their bitmasks. This is implemented by partitioning AVBs by type identifiers, applying a plane-sweeping algorithm to their defined ranges and comparing the bitsets of overlapping AVBs. In this context, the optimizer could potentially materialize a large number of AVBs in memory, many of which can never produce a result.

This is also true for our example query from Figure 5.6. Here, the two dispatch operators that return AVBs for the triple pattern *(?c formula ?f)* might produce large intermediate results, as the pattern is highly unselective. On the other hand, only a small fraction of these bindings will join with the results of the highly selective pattern *(?c contraindication Hypovolemia)*. The same is true for the rightmost dispatch operator. Preventing the materialization of these redundant intermediate results is very important in order to reduce join complexities and thus execution times.

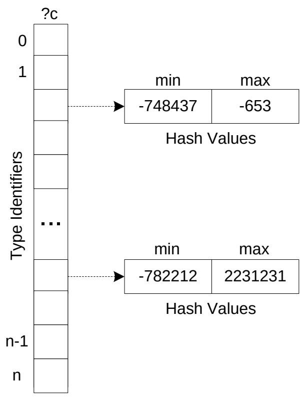


Figure 5.8: Data structure for SIP

As our synopsis includes comprehensive type information, a very light-weight SIP technique can be implemented. To this end, we simply create one array for each join variable, in which we keep track of all type identifiers that could still find a join partner. These arrays are updated whenever an operator has finished its execution. The first operator that produces a complete set of bindings for a join variable initializes the array by setting a pointer to the minimum and maximum hash value encountered for any AVB with this type identifier (see Figure 5.8). All operators that produce a complete set of bindings for the same variable in subsequent steps update the global array to reflect their intermediate result. In our synopsis, type identifiers are stored in the partition table for each PARTree. Whenever a dispatch operator is executed, it can therefore utilize the array to exclude a large number of partitions in a very efficient manner. Moreover, the minimum and maximum hash values are utilized to prune redundant variable bindings while traversing PARTrees during an index scan.

When the optimizer performs join ordering, it orders the operators in a way that guarantees that the estimated costs for processing the left subtree of an operator are lower than (or equal to) the costs for the right subtree. During query simplification, the optimizer performs a post-order traversal of the query execution plan, effectively resulting in a greedy lowest-local-costs-first strategy. This ensures that sideways information passing will rapidly restrict the number of valid bindings to a variable if the execution plan includes highly selective subtrees.

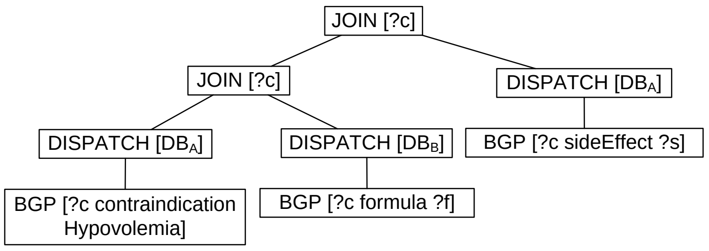


Figure 5.9: Simplified query execution plan

Figure 5.7 shows how sideways information passing can be utilized when optimizing our example query. The dispatch operator with a constant object (*Hypovolemia*) will presumably be rather selective and return just a few AVBs. In contrast, the other dispatch operators with variable subjects and objects and will not be very selective. When the selective dispatch operator has been executed, it initializes the global array which restricts the set of potential bindings for the variable *?c*. This information is first utilized by the two dispatch operators in the lower left subtree to exclude many partitions and PARTree nodes when scanning the index. When the left join has been executed, the array is updated and now reflects the intermediate result of the join operator. This information is then passed to the rightmost dispatch operator, in order to reduce the number of its intermediate results. As can be seen in the evaluation in Section 5.5, this mechanism speeds up query optimization by large margins.

5.3.3 Post-processing

In the post-processing phase, the resulting operator tree is transformed and subqueries are derived. If the set of resulting AVBs is empty, the query can not return any results and query execution can therefore be omitted. Otherwise, there is a potential to prune redundant dispatch-operators that have not contributed to the result. To this end, we monitor the resulting AVBs and keep only those operators who have contributed to at least one AVB. The others are pruned from the query execution plan.

For our example query from Figure 3.5 we assume that the results for the triple pattern `(?compound, formula, ?formula)` from source DB_C did not contribute to the result. Figure 5.9 shows the execution plan after pruning the redundant operator.

Afterwards the tree is reorganized in order to push down joins between triple patterns behind one common dispatch operator whenever possible. The triple patterns of these operators can then be unified to form a more complex basic graph pattern.

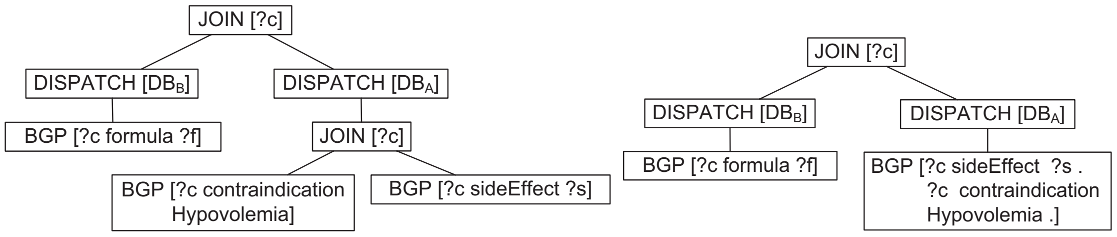


Figure 5.10: Reorganization and pushdown of operators

As in [HAR11] we rely on a heuristic assuming that the minimal number of subqueries yields optimal performance. In a first step, we apply the concept of *Exclusive Groups* [SHH+11]. An exclusive group is a set of subplans that exclusively join with each other and can thus be merged into a common subquery. Analogously, we call subplans that are hidden behind a union operator a *Union Group*. Operators from union groups can not be merged, as any of the intermediate results from an operator in one group might join with an intermediate result from all operators in the other groups. The Figures 5.10 and 5.11 show the transformed execution plan as well as the derived subqueries for our example query.

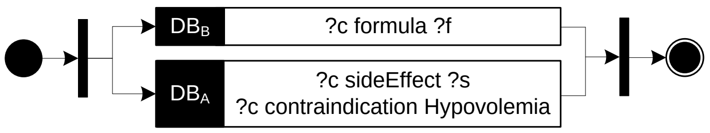


Figure 5.11: Subqueries for the example query

For executing queries, we rely on a simple query execution model that will be explained in more detail in the following section. As is shown in Figure 5.11, we execute all subqueries in parallel while materializing the intermediate results in a temporary database. Finally, the global result is computed by evaluating the original query against the temporary database. Due to the low volume of intermediate results, this process does not introduce a significant overhead when computing the complete result set of a query. However, it does not provide early results during query execution. On the upside, it offers a way to describe the proposed optimizations in a more intuitive manner without the need to present complex transformations to the global query execution plan. All described techniques can also be applied to more complex query processors, as, e.g., any plan similar to the one shown in Figure 5.11 can be translated into a traditional query execution plan.

5.3.4 Merging Query Groups

With the presented process, the optimizer obtains in-depth knowledge about the results of a global query and its subqueries. For some queries, this information can be utilized to further reduce the number of subqueries. An example is shown in Figure 5.12. As can be seen, the query contains a highly unselective triple pattern, which can be answered by any data source in the federation.

```
SELECT ?predicate ?object
WHERE {
  ?subject      ?predicate      ?object .
  ?subject      sameAs          BarackObama .
}
```

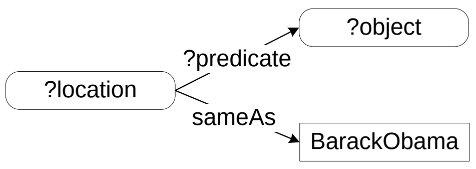


Figure 5.12: Example query 2

The simplified execution plan is shown in Figure 5.13. We assume that the optimizer successfully ruled out most of the potential sources for the contained triple patterns.

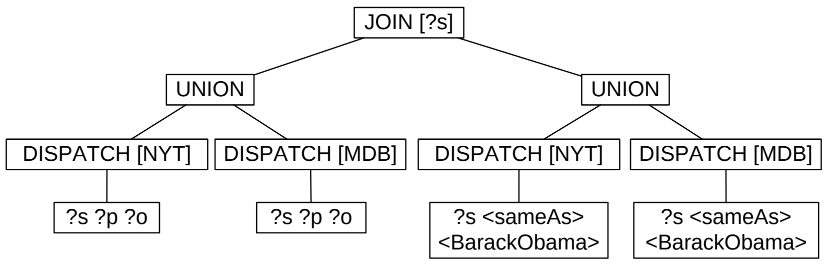


Figure 5.13: Simplified execution plan for query 2

Current query optimizers would derive one subquery for each dispatch operator in each *Union Group*, as any of the intermediate results from an operator on the left side might join with an intermediate result from both operators on the right side and vice versa. In our context, we can utilize the information about which operator contributed to which approximate variable binding to further merge some subtrees.

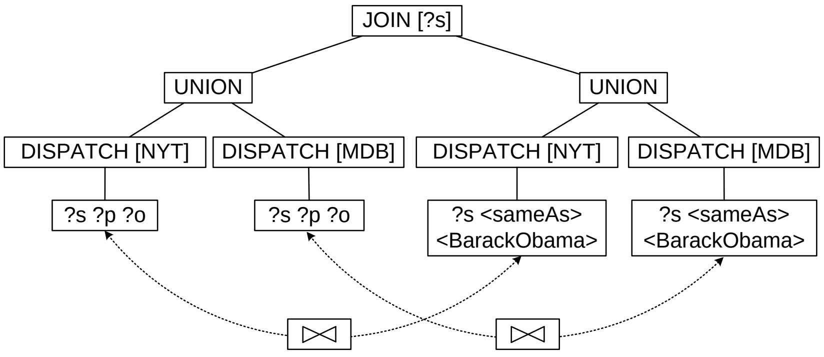


Figure 5.14: Exclusive joins between union groups

As is exemplified in Figure 5.14, it might be the case that some operators from different groups exclusively join with each other. It is then possible to reorganize the execution plan and to derive less subqueries without sacrificing result completeness. For a query Q, the set of all pairs of joining dispatch operators is given by:

$$\mathit{JOINS}(Q) = \{\{d_1, d_2\} \mid \exists_{b \in AVB(Q)} \; : \; d_1 \in b.\mathit{dispatches} \wedge d_2 \in b.\mathit{dispatches}\}$$

We further assume, that *OPS(Q)* returns the set of dispatch operators contained in any query or subquery Q. In contrast to the rather simple example shown in Figure 5.14, elements from union groups can also be merged partially.

The pseudocode of a basic algorithm is shown in Algorithm 4, which requires a set of query groups G which contains all union groups as well as all merged exclusive groups for a query Q as input. For each pair of joining groups it determines all pairs of the contained queries that join exclusively with each other. These subqueries are then removed from their groups, combined with each other and added to a new group. If a new group is created, the whole process is repeated in order to potentially perform merges with the new, more complex, queries. All remaining queries from an union group in which only some of the subqueries could be merged, are removed from the overall process, as they can not be merged any more without sacrificing consistency. However, they are still contained in the final set of subqueries.

For the optimizer, merging query groups comes at no significant additional costs as it solely relies on information that is computed anyway during query optimization. On the other hand, each merge of two operators from two query groups results in one less subquery to be evaluated. Performing all merges for the groups in our example query results in the execution plan shown in Figure 5.15.

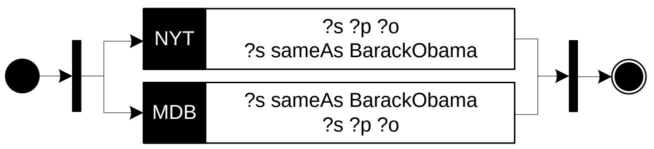


Figure 5.15: Subqueries for query 2

Algorithm 4: MergeQueryGroups

```
Input: Set of query groups $G$, Query $Q$
Input: New set of query groups $G'$
begin
    $G' \leftarrow \emptyset$
    foreach joining pair $(g_1, g_2) \in (G \times G) \mid g_1 \neq g_2$ do
        /* Determine pairs of joining subqueries */
        $j \leftarrow \emptyset$
        foreach subquery $q_1 \in g_1$ do
            foreach subquery $q_2 \in g_2$ do
                $o_1 \leftarrow \text{OPS}(q_1)$
                $o_2 \leftarrow \text{OPS}(q_2)$
                if $\exists_{(d_1,d_2) \in \text{JOINS}(Q)} | d_1 \in o_1 \wedge d_2 \in o_2$ then
                    $j \leftarrow j \cup \{(q_1, q_2)\}$
        /* Merge queries with only one join partner */
        $H \leftarrow \emptyset$
        foreach $(q_1, q_2) \in j$ do
            if $\nexists_{(q_1,x) \in j} x \neq q_2$ and $\nexists_{(y,q_2) \in j} y \neq q_1$ then
                $H \leftarrow H \cup \{q_1 \bowtie q_2\}$
                /* Remove from exclusive groups */
                $g_1 \leftarrow g_1 \setminus \{q_1\}$
                $g_2 \leftarrow g_2 \setminus \{q_2\}$
        /* If merge was performed */
        if $H \neq \emptyset$ then
            /* Result contains rest of $g_1$ and $g_2$ */
            $G' \leftarrow G' \cup \{g_1, g_2\}$
            /* Replace original groups with $H$ */
            $G \leftarrow G \setminus \{g_1, g_2\}$
            $G \leftarrow G \cup \{H\}$
            /* Start over */
            goto 3
    /* Kepp remaining groups */
    $G' \leftarrow G' \cup G$
    return $G'$
```

5.4 Query Execution

In the query execution phase, the subqueries are sent to the local databases and the global result is computed by processing the local results. In this context, it is possible to implement further run-time optimizations. We first present a novel technique for pruning redundant intermediate results, which forms the backbone of our query processor and aims at reducing network communication to a minimum. We then present several further approaches that are utilized to increase the selectivity of subqueries. Our optimizer incorporates these in a dynamic manner and is the first to utilize a portfolio of different query execution mechanisms, depending on query characteristics. Finally, the overall architecture of our querying engine is described in more detail.

5.4.1 Pruning Redundant Intermediate Results

Our query optimizer allows to implement several techniques for reducing the volume of intermediate results during query execution. Firstly, this includes generic approaches, such as the bind join mechanism [Kos00] which our optimizer is able to apply selectively, i.e., whenever it is potentially useful. Secondly, this includes techniques that are specific to our approach, such as replacing unbound predicates with constants and pruning redundant intermediate results by utilizing an approach related to semi-join reducers.

An important side-effect of the presented approach is that the AVBs resulting from query optimization can also be used to prune redundant variable bindings during query execution. As AVBs describe restrictions for hash values of variable bindings they can be used to filter a stream of bindings, leaving only those who are potentially relevant for answering a query. A set of n-dimensional AVBs can be derived from a set of bindings B for n variables by iteratively hashing the values of each binding and approximating them in the same way as during index generation, i.e., by applying the Sort-Tile-Recursive algorithm. When we denote this by a function $AVB(B)$ and returning bindings for a query execution plan P by a function $Q(P)$ the following holds for a query consisting of a join between two triple patterns T_1 and T_2:

$$AVB(Q(T_1 \bowtie T_2)) \approx AVB(Q(T_1)) \bowtie AVB(Q(T_2))$$

As $AVB(Q(T_1)) \bowtie AVB(Q(T_2))$ is exactly what is computed by the optimizer during query simplification, the AVBs returned by the optimizer effectively approximate the variable bindings that would be returned when executing the query. This is true for any other query and can be used to implement a concept related to semi-join reducers (see, e.g., [Kos00, SKBK01]). In this approach irrelevant join candidates for a join between two relations from remote systems are pruned by matching them against the results of a semi-join operation. Although there are different ways to implement semi-join reducers (e.g., with bloom-filters), they have in common that the reducers are computed at run-time whereas in our case they are generated at compile-time.

The optimizer returns a set of n-dimensional AVBs (one dimension for each variable in the query) but individual subqueries normally only cover parts of these variables. We therefore split each resulting AVB into multiple (potentially overlapping) AVBs, one for each set of variables contained in a subquery. As the set of AVBs for an individual subquery might now contain duplicates, we further apply a distinct operator. In order to efficiently check variable binding against the reducers, we again organize AVBs in PARTrees by applying the Sort-Tile-Recursive algorithm. In contrast to indexing where we build two-dimensional trees, the number of dimensions now depends on the number of variables in the subquery. Moreover, we do not partition the resulting AVBs by data types, in order to keep the space overhead low. Instead, we organize them in one single PARTree in which nodes also store type information. For each dimension, the data type of a node is defined as the result of a bitwise OR-operation on the according type identifiers of its children. This is inspired by S-Trees, a variant of which has also been used for indexing RDF datasets in [ZMC$^+$11].

The trees generated by this process are then transferred to the remote systems together with the subqueries. During the execution of a subquery, members of the stream of resulting variable bindings are pruned if there is no leaf node in the reducer matching the binding's types and hash values. This does not affect the completeness of the overall result, because the generated reducers only allow for false positives.

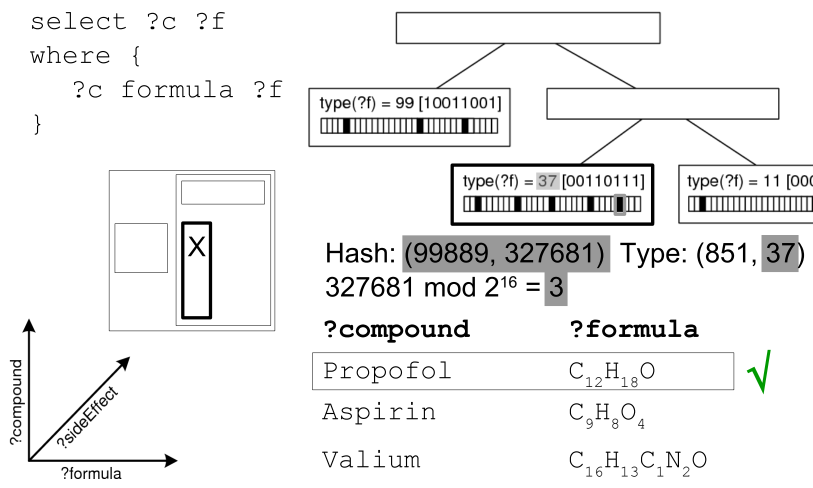


Figure 5.16: Example for applying a reducer

This process is exemplified in Figure 5.16. It shows how the second subquery from Figure 5.11 is evaluated at a knowledge base about compounds and their chemical properties. The generated reducer is a two-dimensional spatial tree structure, which covers the *compound* and *formula* dimensions. The leaf nodes of this tree contain one bitmask for each of these dimensions, of which only one is shown in the figure.

The query is executed at the local database and for each of the resulting variable bindings it is checked whether they are a valid result according to the associated reducer. In this example, we check the binding (*?compound=Propofol*, *?formula=*$C_{12}H_{18}O$). We assume that the two-dimensional point derived by hashing the constants is *(99889, 327681)* and the associated type information is *(851, 37)*. Firstly, when checking the binding for validity, the hash values are utilized to traverse the tree structure up to a leaf node. If no leaf node is found, the binding is not valid. In the example we assume that there is a valid leaf node, which is highlighted in the figure. Secondly, the type information is utilized to check whether the according bit is set in the bitset related to each dimension. In the example, we assume that this is true for both dimensions and the binding is a valid local solution for the subquery. This means, that it is very likely to contribute to the overall result of the query.

5.4.2 Cardinality Estimation

In the remainder of this section, we will present several advanced execution mechanisms for distributed SPARQL queries. To choose the most appropriate techniques for a given query, the optimizer utilizes cardinality estimates. We assume that for any query Q, *AVB(Q)* returns the approximated result, *VAR(Q)* returns all variables in Q and *OPS(Q)* returns the identifiers of all dispatch operators contained in Q. When computing the reducer $AVB(Q', Q)$ for a subquery Q', we extract the relevant subsets of *AVB(Q)* and apply a distinct operator to remove duplicates caused by joins with other subqueries:

$$AVB(Q', Q) = \{(b.type_v, b.min_v, b.max_v, b.bitset_v, ...) \\ \forall v \in VAR(Q') \text{ and } b \in AVB(Q) \ : \ OPS(Q') \subseteq b.dispatches\}$$

The approximate bindings *AVB(Q)* that result from optimizing a query Q and the reducers *AVB(Q', Q)* for each subquery Q' can also be utilized for accurate cardinality estimations. For a set of AVBs B over a set of variables V we derive an estimation of the cardinality of the approximated bindings in the following manner:

$$CARD(B) = \sum_{b \in B} \max(\{|b.bitset_v| \ : \ v \in V\})$$

As can be seen, the basic idea is that the number of active bits in an AVBs bitsets represents the number of approximated bindings. As there might be collisions between different values when building the bitsets during an index scan, we determine the maximum number of active bits for any dimension. We add up the numbers of all AVBs in order to derive an estimation of the overall cardinality.

With these preliminaries we are able to define two functions that return the estimated reduced cardinality $C_{REDU}(Q', Q)$ and the estimated original cardinality $C_{ORIG}(Q')$ for any subquery Q' of a global query Q. The reduced cardinality estimates the size of a subquery's intermediate result after applying the reducer:

$$C_{REDU}(Q', Q) = CARD(AVB(Q', Q))$$

In contrast, the original cardinality is an estimation of the cardinality of a subquery's result when executed without applying a reducer. It can be computed in an additional optimization phase in which the subquery is treated as an individual query which is to be executed at one predefined data source. This allows to retrieve a set of AVBs $AVB(Q')$ that does only approximate the actual results of the query and does not incorporate any restrictions induced by joins with other subqueries. The original cardinality is defined as:

$$C_{ORIG}(Q') = CARD(AVB(Q'))$$

Although the query execution model with compile-time generated reducers minimizes the volume of intermediate results, it is not possible to leverage this optimization for speeding up local query execution. This means that if a subquery is not very selective, many tuples are extracted from the local database and then pruned by applying a reducer. This process is potentially time consuming, as is exemplified by the query shown in Figure 5.17. We assume that the first, selective, triple pattern of this query is to be executed at another source than the second, highly unselective, triple pattern. In this case, a query would be executed that selects all triples stored in a database.

```
SELECT ?predicate ?object
WHERE {
  DB00201      sameAs        ?caff .
  ?caff        ?predicate    ?object .
}
```

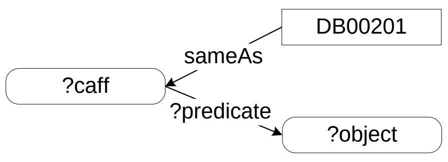


Figure 5.17: Example query 3

To solve this problem and decrease the query execution time for queries with unselective subsets, we present two approaches in this section. For both techniques, the optimizer relies on cardinality estimates. If beneficial, the optimizations are incorporated into the execution plan and executed at runtime.

In general, it only makes sense to further optimize a subquery, if it is highly unselective, but its reduced cardinality is very low, i.e., a large fraction of the resulting tuples will be filtered by a reducer. If this is the case, an increased selectivity of a subquery can potentially result in a significant reduction of the complexity of local query processing. When the optimizer sees a potential to apply further optimizations, which will be explained in more detail in the following sections, it marks a query Q' as unselective if its original cardinality is larger than a threshold:

$$C_{ORIG}(Q') \geq c_{unselective}$$

For unselective subqueries, further optimizations can only be beneficial if a large portion of the resulting tuples will be filtered by a reducer, which is expressed as:

$$\frac{C_{REDU}(Q')}{C_{ORIG}(Q')} \leq c_{ratio}$$

5.4.3 Predicate Binding

To increase the selectivity of queries with unbound predicates, our optimizer employs a technique which we call *predicate binding*. In contrast to information about the triples' subjects and objects, our data summary does not approximate information about predicates but preserves it for each partition. During query optimization, the optimizer determines valid data sources, type identifiers and boundaries for all potential bindings to all variables, including predicates.

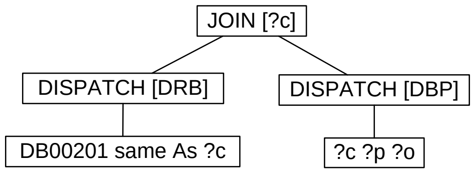


Figure 5.18: Simplified execution plan for query 3

When the set of predicates contained in the data sources is known, the optimizer can utilize this information to compute a set of resources that are valid bindings for variable predicates. This set is defined by all predicates with a type identifier and hash value that match any of the resulting AVBs. The set of bindings for a variable predicate is then sent to the data source together with the query. During local query execution, these bindings are inserted into the subquery in order to increase its selectivity. The set of predicates contained in each subsystem can be determined during indexing, without any additional runtime overhead.

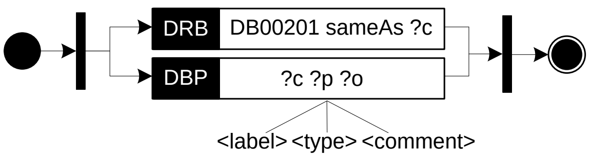


Figure 5.19: Bound predicates for query 3

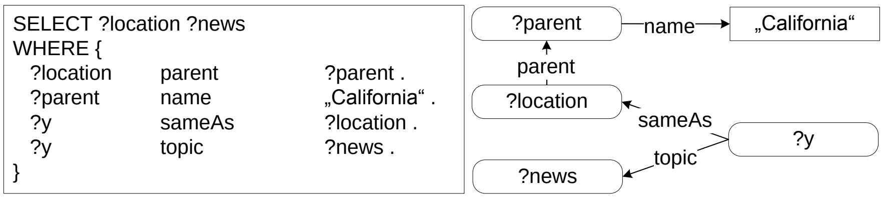
Assuming that the set of predicates of a data source S is given by *PRED(S)* the potential bindings for a variable predicate $v \in$ *VAR(Q')* of a subquery Q' of a query Q that is to be executed at S is defined as:

$$\begin{aligned} \mathit{BINDINGS}(v, S, Q', Q) = \{p \in \mathit{PRED}(S) \mid \exists_{b \in AVB(Q',Q)} :& \\ \mathit{type}(p) = b.\mathit{type}_v \wedge& \\ \wedge\ b.\mathit{min}_v \leq \mathit{hash}(p) \leq b.\mathit{max}_v \wedge& \\ \wedge\ (\mathit{hash}(p) \mod 2^{16}) \in b.\mathit{bitset}_v\}& \end{aligned}$$

To prevent the potential costs of additional optimization phases, we compute the original selectivity only for subqueries with unbound predicates. An execution plan with bound predicates for our example query from Figure 5.18 is shown in Figure 5.19.

5.4.4 Bind Joins

Bind joins are a common technique implemented in all state-of-the-art distributed query processors for RDF databases, such as the systems presented in [LWB08, QL08] and [SHH+11]. The basic idea is to compute a join between two subqueries by sequentializing their execution and binding a variable in one query to all according values obtained by executing the other query. For previous systems, the bind join mechanism is the most important technique for reducing the volume of intermediate results. To this end, the bind join is simply performed whenever possible, while row-blocking is utilized to allow for parallelized processing.



```
SELECT ?location ?news
WHERE {
  ?location     parent       ?parent .
  ?parent       name         „California“ .
  ?y            sameAs       ?location .
  ?y            topic        ?news .
}
```

Figure 5.20: Example query 3

Similar to predicate binding, bind joins are relevant for speeding up local query processing in our system. An example is shown in Figure 5.20. We assume that optimizing this query results in the simplified query execution plan shown in Figure 5.21. The execution plan consists of five query groups, which results in the execution of five different subqueries. While some of these queries are rather selective (with bound objects), others are highly unselective.

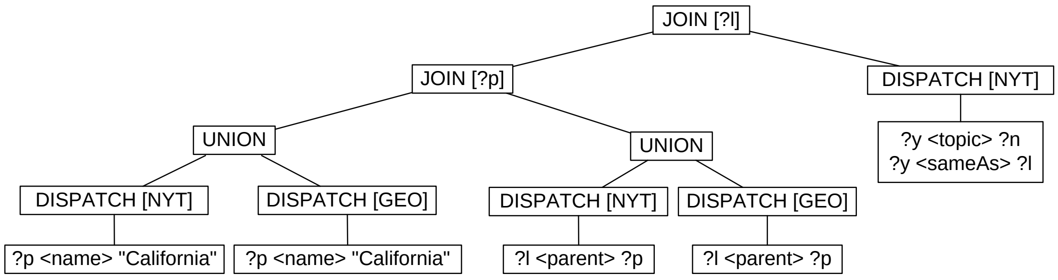


Figure 5.21: Simplified execution plan for query 3

In contrast to previous approaches, our optimizer does not solely rely on the bind join operator, but utilizes cardinality estimates to execute a bind join whenever it seems appropriate. First, we determine whether any of the subqueries or any subset of the subqueries is a potential *source* for a bind join. Here, it is important to make sure that a source provides all possible bindings for its triple patterns. This means that a source can also be a set of subqueries, if more than one query returns bindings for a contained pattern. With one exception that will be explain later in this section, these groups resemble the query groups from the previous section. A query group $G = \{Q_0, ..., Q_i\}$ can only be a source if it is highly selective, i.e., if its cumulated reduced cardinality lies under a predefined threshold:

$$C_{REDU}(G) = \sum_{Q_i \in G} C_{REDU}(Q_i) \leq c_{source}$$

Note that the set of potential sources can be determine without a significant overhead, as it solely relies on the results of query optimization. If a query execution plan does not include any subqueries that are valid sources, we can therefore skip any additional optimization phases that would be required to determine the estimates of the original cardinalities of potential targets.

Algorithm 5: OPTIMIZEJOINGRAPH

```
Input: Initial join graph $G = \{e = (src, var, grp, dst)\}$
begin
    /* Retain sources */
    foreach $e \in G$ do
        if $C_{REDU}(e.grp) \geq c_{source}$ then
            $G \leftarrow G \setminus \{e\}$
    /* Retain targets */
    foreach $e \in G$ do
        if $C_{ORIG}(e.dst) \leq c_{unselective}$ or $\frac{C_{REDU}(e.dst)}{C_{ORIG}(e.dst)} \geq c_{ratio}$ then
            $G \leftarrow G \setminus \{e\}$
    /* Remove redundant dependencies */
    foreach $(e_1, e_2) \in G \times G \ : \ e_1.dst = e_2.dst \land e_1.grp \neq e_2.grp$ do
        if $C_{REDU}(e_1.grp) < C_{REDU}(e_2.grp)$ then
            $G \leftarrow G \setminus \{(src, var, e_2.grp, e_2.dst) \in G\}$
        else
            $G \leftarrow G \setminus \{(src, var, e_1.grp, e_1.dst) \in G\}$
    /* Remove cycles */
    foreach cycle $\{e_1, e_2, ...e_n\} \in G$ do
        $max \leftarrow e_i \ : \forall_{1 \leq i,j \leq n} C_{REDU}(e_i.grp) \geq C_{REDU}(e_j.grp)$
        $G \leftarrow G \setminus \{(src, var, max.grp, max.dst) \in G\}$
```

A basic algorithm for introducing bind joins into our query execution plans is presented in Algorithm 5. As input, it requires a complete join graph G including all subqueries. If a subquery *src* from a query group *grp* shares a variable *var* with another subquery *dst* we introduce a directed edge reaching from *src* to *dst* that is labelled with *grp* and *var*. In the algorithm this is modelled as a set of tuples of the form *(src, var, grp, dst)*. We further assume that *size(grp)* returns the number of queries in the exclusive group *grp*. The initial join graph for our example query from Figure 5.21 is presented in Figure 5.22.

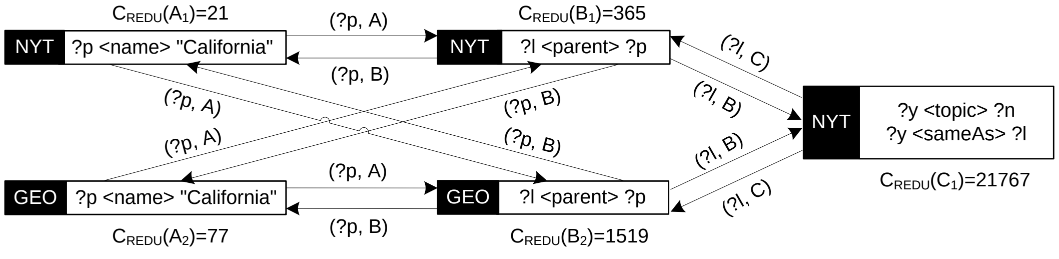


Figure 5.22: Step 1 - Initial join graph

For our example, we assume that $c_{unselective} = 10^5$, $c_{ratio} = 0.05$ and $c_{source} = 2 * 10^3$ which are also the parameters utilized in our experiments in Section 5.5. In a first step, we remove all edges that originate from queries that are not selective enough to be the source of a bind join. This results in the join graph shown in Figure 5.23.

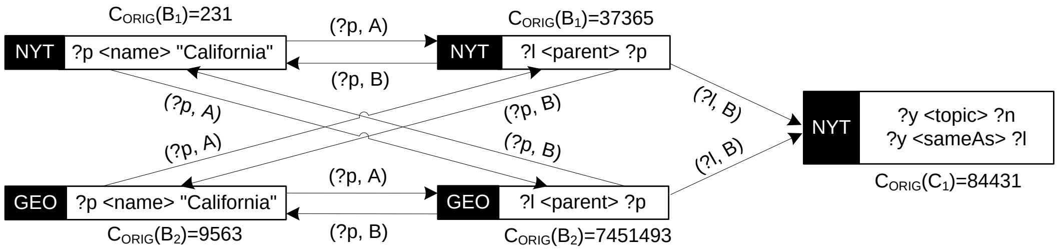
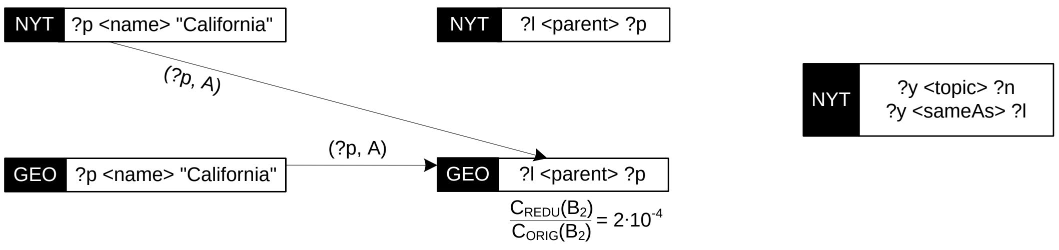


Figure 5.23: Step 2 - Retaining sources

If there is at least one subquery which is a potential source for a bind-join and other subqueries exist which share common variables, we determine the set of potential targets. A target is a subquery that shares at least one variable with a potential source and for which the preconditions defined in the beginning of this section hold true, i.e., it is highly unselective, but its reduced cardinality is very low.



NYT ?p <name> "California"
NYT ?l <parent> ?p
NYT ?y <topic> ?n ?y <sameAs> ?l
(?p, A)
(?p, A)
GEO ?p <name> "California"
GEO ?l <parent> ?p
C_REDU(B2) / C_ORIG(B2) = 2·10^-4

Figure 5.24: Step 3 - Retaining targets

As shown in Figure 5.24 we assume that for our example query most of the queries in the execution plan will not fulfil these conditions. This is also true for real-world queries as these often contain only a few rather selective subqueries. In our example, the optimizer could now start to derive an execution plan from the resulting join graph. However, in the general case it is also important to make sure that the resulting join graph does not result in inconsistencies and does not contain cycles.

As is shown in Algorithm 5, we first make sure that any subquery is not bound by variables obtained from different query groups. If a query would receive bindings for the same variable from different groups it would make more sense to execute a chain of bind joins, because the query would only be required to be bound to values contained in the intermediate results of both groups. If a query would receive bindings for different variables from different groups it would be required to bind it to the cross-product of the values obtained for all variables which is potentially very costly. In these cases, we therefore only retain the binding to the exclusive group with the lowest cumulated cardinality. The same is also true for queries for which predicate binding as well as a bind join to a variable subject or object could be performed. As it would be required to bind the query to the cross product of these values, we only apply the bind join in this case. The reason for this is that predicate binding inserts schema-level information whereas bind joins insert instance-level information which is generally more selective.

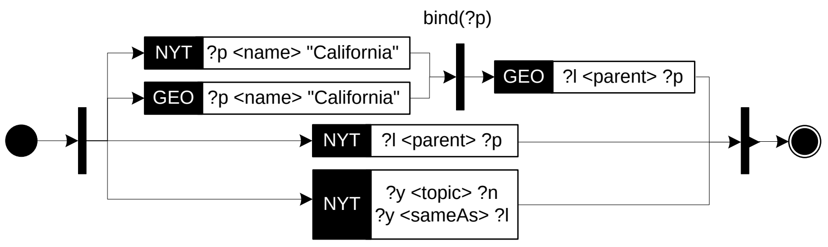

bind(?p)
NYT ?p <name> "California"
GEO ?p <name> "California"
GEO ?l <parent> ?p
NYT ?l <parent> ?p
NYT ?y <topic> ?n ?y <sameAs> ?l

Figure 5.25: Execution plan for query 3

Secondly, we check whether the join graph contains cycles, as a cyclic execution of bind joins is not possible. For each cycle, we remove the dependencies resulting from the query group with the largest cumulated cardinality. From the resulting join graph, an execution plan can be derived by applying topological sorting [CSRL01]. Here, each synchronization point represents a bind join at which the querying engine has to make sure that the subsequent query is bound to all values obtained by executing the previous queries. Queries which are neither a source, nor a target can be executed in parallel to the sequences that implement bind-joins. Our execution plans include all possible bind joins between any compatible sets of sources and targets. The query execution plan for our example join graph is shown in Figure 5.25.

The presented process is slightly simplified, as it assumes that each query does belong to exactly one query group. In the general case, if partial merges of query groups have been performed during query optimization, these queries can belong to different query groups at the same time. This can be considered in the proposed algorithm but has been omitted for the sake of clarity.

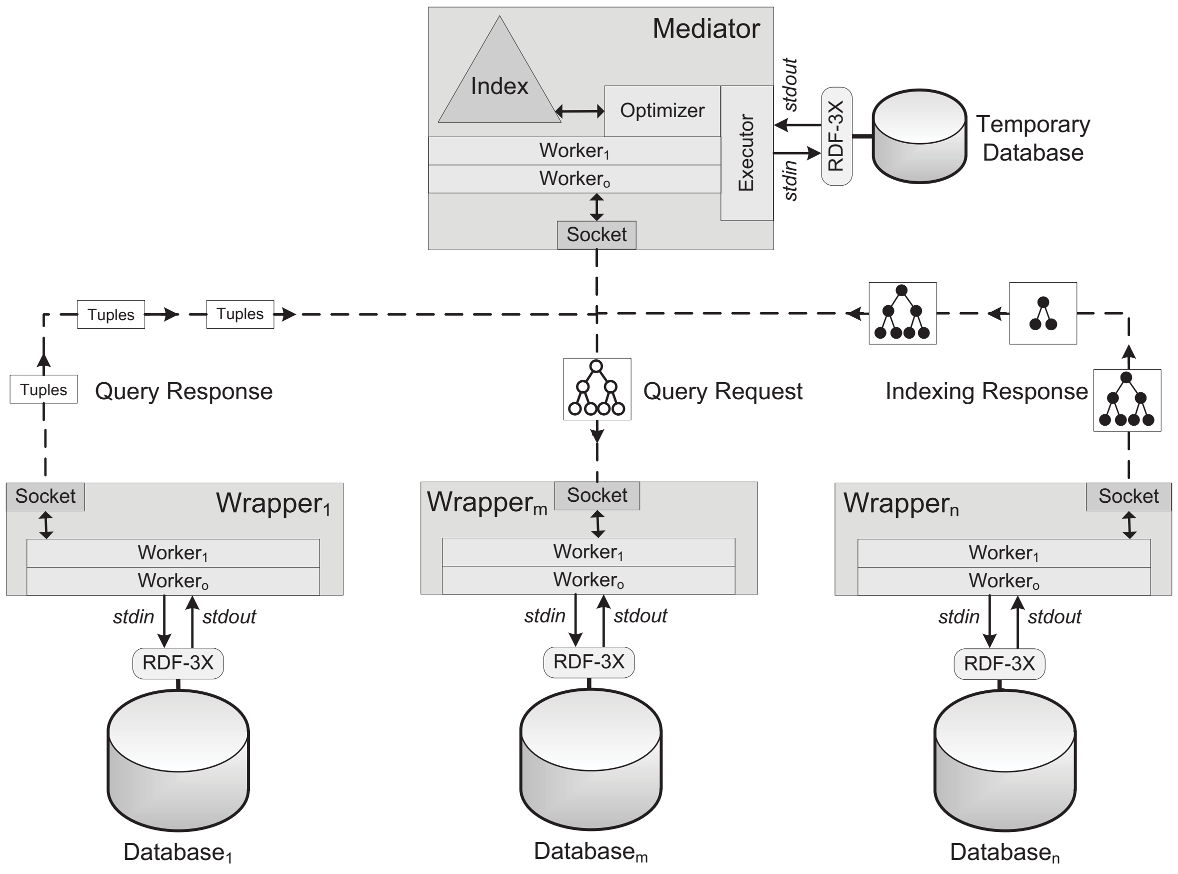


Figure 5.26: System architecture

5.4.5 System Architecture

In order to evaluate our approach we have developed a query engine based on the mediator/wrapper architecture shown in Figure 5.26. It implements a query execution model which consists of four steps:

1. Parse and optimize the query.
2. Execute subqueries at the remote systems (partly sequentialized with bind joins).
3. Load local results into a global database.
4. Execute the query on the global database.

This is basically equivalent to executing the query on a relevant subgraph of the global graph. Therefore an RDF store can again be used as a global temporary database. Our prototype implements all concepts described in this chapter. It supports distributed indexing as well as query optimization and execution, as is shown in Figure 5.26.

Wrappers export standardized interfaces to the remote systems which are provided by instances of the RDF-3X database system [NW10]. As the wrappers are loosely coupled to the underlying databases and only require a SPARQL interface, we are able to integrate nearly any RDF store. We chose RDF-3X because it is one of the most efficient, open-source RDF database systems available and offers excellent performance. All components are multithreaded and able to process several queries in parallel. Mediator and wrappers communicate via messages exchanged over plain sockets. The temporary global database is also implemented as an instance of the RDF-3X system. Because the developed optimizations generally cut down heavily on the number of intermediate results (see section 5.5) it is also possible to replace it by an in-memory database. Unfortunately the number of intermediate results is much higher without optimizations and they do thus often not fit into main memory. We therefore chose RDF-3X to allow for a fair comparison of the unoptimized and optimized case in the following section.

5.5 Evaluation

There is currently no established benchmark for distributed SPARQL processors. Although several benchmarks for stand-alone RDF stores have been proposed [BS09, SHLP09, GPH04], they naturally do not reflect the specific challenges faced by distributed query processors for RDF (e.g. data localization or distributed join processing). *FedBench* is a benchmark for federated SPARQL query processing, which is in an early stage and has not yet been widely adopted [SGH$^+$11]. It does not offer means for a systematic evaluation of a query processor, but provides a valuable setup for comparing different systems.

We therefore start by systematically evaluating the performance of our most important query execution mechanism - compile-time generated reducers - for increasing query complexity and different types of data distribution. This covers a broad spectrum of the challenges faced by federated RDF database systems. To this end, we have generated a synthetic workload based on a collection of biomedical knowledge bases.

Finally, we present an evaluation with *FedBench* and compare our system with other approaches. Here, we also focus on further compile-time and run-time optimization techniques and the performance of our optimizer.

In both experiments, our testbed consisted of three Dell desktops which hosted the data sources and wrappers. Each of these machines has a 4-core 3.1 GHz Intel Core i5 CPU with 6 MB cache and 8 GB of memory running a 64-bit Linux kernel in version 2.6.35. The mediator was deployed on the same machine which has been utilized in the experiments in the previous section, i.e., a Dell laptop with a 4-core 1.6 GHz Intel Core i7 CPU with 6 MB cache and 4 GB of memory running a 64-bit Linux 2.6.35 kernel. All systems are able to perform sequential reads on their local hard disks with about 100 MB/s and were connected via Fast Ethernet. Mediator and wrappers are implemented in Java and all machines were running a 64-bit Sun JVM in version 1.7.0. The heap size was restricted to 512 MB for each wrapper and 3 GB for the mediator.

5.5.1 Evaluation of the Pruning Power of our Approach

Datasets

Our evaluation datasets contained roughly 100 million triples which were distributed over 10 databases. The scenario models a knowledge base for drug developers and researchers in the area of medicine, and incorporates the interlinked RDF databases presented in Section 4.5. An overview over the characteristics of the datasets is shown in Figure 5.1

System	# Triples	# Subjects	# Predicates	# Objects
Infobox Properties	34.2 M	1.816.862	38.563	8.107.107
Other Properties	31.3 M	9.490.850	8	13.590.111
GeneID	20.1 M	462.855	31	10.750.501
Linked CT	9.8 M	981.880	90	3.808.369
HGNC	1.1 M	125.256	37	655.833
OMIM	0.9 M	20.280	43	379.099
Drugbank	0.5 M	19.693	119	275.336
Dailymed	0.2 M	10.015	28	67.778
Sider	0.1 M	2.674	11	29.410
Diseasome	0.1 M	8.152	19	27.704
Global	98.4 M	11.121.647	38.905	35.837.311

Table 5.1: Characteristics of the evaluation datasets

We evaluated three different scenarios by partitioning the dataset in different ways. In the *naturally-partitioned* scenario the datasets have been preserved as is. For the *horizontally-partitioned* scenario we merged all datasets into one single dataset and partitioned it horizontally into n equally sized datasets. In the *randomly-partitioned* scenario all triples have been distributed randomly among n equally sized datasets. These three scenarios model different ways in which we expect data to be collected. The rationale for natural data distribution is straightforward as this models the case in which several subject-specific data collections have been established. Horizontal data distribution resembles a scenario in which different knowledge bases have been built by merging subsets of other datasets. Finally the randomly-distributed scenario models an extreme for an RDF-specific type of data collection. We assume that users take advantage of their ability to uniquely reference entities in other datasets and further annotate them. Therefore the information regarding individual entities is spread over different data sources. A real-world data management solution for our biomedical use cases would probably have to deal with a data distribution lying somewhere in between these three scenarios.

When partitioned naturally, we distributed the datasets among the three machines as follows:

- **Machine A**: Infobox Properties, OMIM, Drugbank (≈36 M triples)
- **Machine B**: Other Properties, HGNC, Sider (≈32.6 M triples)
- **Machine C**: GeneID, LinkedCT, Dailymed, Diseasome (≈30.2 M triples)

In case of horizontal or random partitioning we derived nine equally sized datasets (≈10.9 M triples) and distributed them uniformly among the three machines (three datasets each).

Workload

The workload used for evaluating the presented approach is generated from different patterns. A query pattern is defined by a number of stars (s), a number of constants (c) and a number of variables (v). Each star defines a basic graph pattern that consists of (c+v) triple patterns. Subjects of these triple patterns are always variables, predicates are always bound and c of the objects are constants, whereas v of the objects are variables. If a pattern consists of more than one star, the individual stars are connected via additional triple patterns with unbound subject and object and bound predicate. Therefore a query pattern with parameters s,c and v consists of $n = (c + v + 1) * s - 1$ triple patterns. Such queries are considered to be good representatives for many real world SPARQL queries [AFMPdlF11] and similar query patterns have been used in other evaluations [AMMH07, NW10, HAR11]. In the following sections we denote a pattern consisting of s stars with v variables and c constants as $S_s V_v C_c$. Figure 5.27 shows an overview over the set of query patterns used in this evaluation. It comprises patterns consisting of one, two and three stars, each of which consists of one, two or three constants and one or two variables. The most complex query pattern ($S_3 V_1 C_2$) consists of $n = 11$ triple patterns.

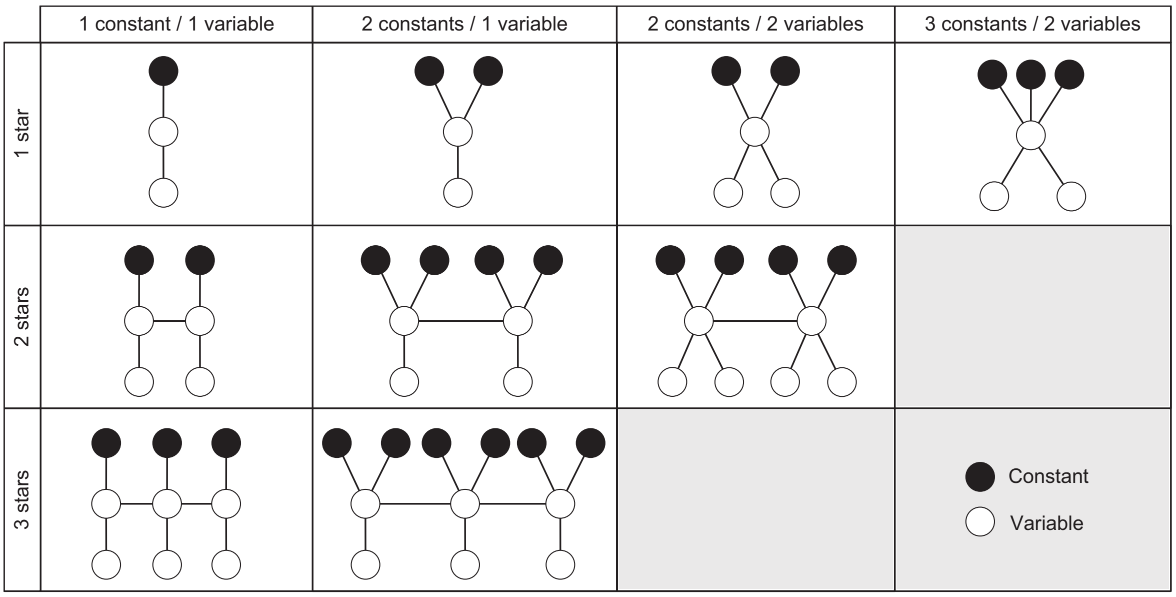


Figure 5.27: Query patterns

We generated a workload by creating roughly 100 instances of each pattern. To this end we associated stars to data sources (naturally-partitioned). A random instance of a star was generated by selecting all possible outgoing triples for a random subject, replacing the subject with a variable, randomly selecting v triples and replacing its objects by variables and c triples that were kept as is. In order to generate a workload which can be executed in a reasonable amount of time without applying any optimizations we ensured that each star does not yield more than 10.000 results. Additionally, we matched links between stars with links between datasets which allowed us to control how the load is spread among the data sources. For each possible link between two (or three) datasets we created an equal number of queries in a way that resulted in about 100 instances. In the case of $s = 1$ this was trivial. As there are 10 data sources we just created 10 random instances for each dataset. For patterns consisting of $s = 2$ stars we were able to discover 25 possible links between two data sources. Therefore we created four random instances per query pattern and link, which resulted in exactly 100 queries. For $s = 3$ this was more complicated as links between data sources are not always transitive. We found that there are 32 possible links between three datasets and created three instances for each of them, resulting in 96 queries per pattern.

Query Class	$\leq 10^1$	$\leq 10^2$	$\leq 10^3$	$\leq 10^4$	$\leq 10^5$
$S_1V_1C_1$	71	5	6	11	7
$S_1V_1C_2$	88	8	1	2	1
$S_1V_2C_2$	82	7	7	3	1
$S_1V_2C_3$	87	9	3	1	0
$S_2V_1C_1$	87	6	4	3	0
$S_2V_1C_2$	98	1	1	0	0
$S_2V_2C_2$	91	6	2	1	0
$S_3V_1C_1$	73	20	3	0	0
$S_3V_1C_2$	88	5	3	0	0
Total	765	67	30	21	9

Table 5.2: Result cardinalities

Although there are no guarantees that this process generates queries that spread the load equally among the different datasets (i.e., query each dataset with the same frequency), it works well in practice. Regarding the naturally-partitioned scenario, the load is spread equally for $s = 1$. In case of $s = 2$ OMIM and GeneID are slightly underrepresented whereas Diseasome and Drugbank are overrepresented. For $s = 3$ OMIM, GeneID and HGNC are slightly underrepresented whereas Diseasome, Drugbank and Dailymed are overrepresented. All other datasets are queried with the same frequency. Table 5.2 shows the result cardinalities of the queries in our workload. Due to limiting the results of the individual stars to a cardinality of $\leq$10.000 most queries ($\approx$85%) return less than 10 results. As can be seen from the following experiments, the overall time needed to execute a query is usually not strongly correlated with the cardinality of its result but with the cardinalities and number of its subqueries which are again influenced by data distribution. As it is more robust against outliers, we report the geometric mean of the running times for the different sets of queries.

Indexing

The size and accuracy of the synopsis can basically be controlled by adjusting the PARTrees' leaf fanout parameter (f_l) as it defines the number of triples that are packed into one leaf node. Figure 5.28 shows the size of the summarization for different values of f_l in the case of natural data distribution. Space consumption converges to a lower boundary when increasing f_l. This is due to the fixed four bytes allocated for each indexed triple. The additional overhead for nodes in the PARTrees decreases with f_l.

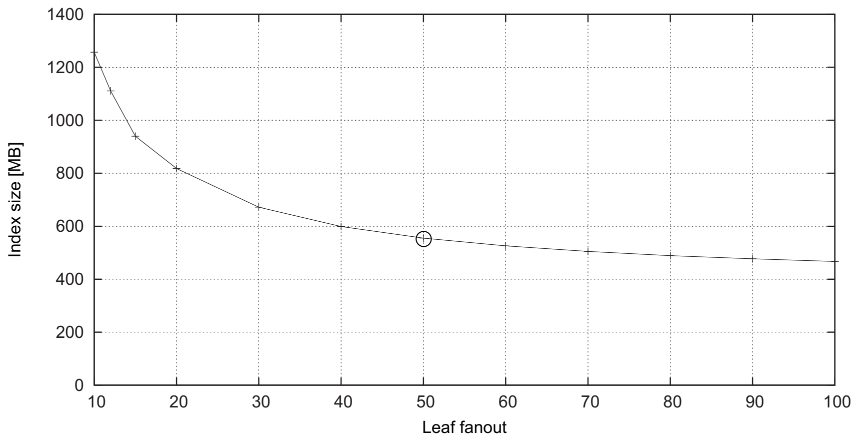


Figure 5.28: Size of the synopsis for natural partitioning

For our evaluation we chose f_l to be 50 which seems to be a good trade-off between space consumption and performance according to our experiments. In this case the synopsis consumes about 5.6 bytes per triple. This totals to about 555 MB of main memory which corresponds to a compression ratio of about 5% compared to the original data in RDF turtle format (12 GB). In case of horizontal and random data partitioning the space consumption increases slightly to 560 MB and 571 MB respectively. Building the summary from scratch takes about 18 minutes for natural data distribution and 10 minutes for the other scenarios. The latter benefit from the fact that each dataset contains an equal number of triples, which leads to a higher level of parallelization. About 30% of the time is spent on building the prefix tree.

Query Optimization

The average time needed to optimize the generated SPARQL queries is shown in Figure 5.29. The measured running times include all optimization steps, ranging from parsing the query and initial plan generation, to plan simplification and the generation of reducers. As can be seen the running time increases with the complexity of the underlying query pattern. The patterns $S_2V_2C_2$ and $S_3V_1C_2$ are the most difficult, as they contain the most variables and joins. Furthermore, the complexity increases slightly when optimizing queries in case of horizontally- and randomly-partitioned data. Analogously to the increasing size of the synopsis this is due to the fact that these partitionings lead to a larger number of unique predicates per dataset and therefore increase the number of partitions. Triples that are summarized in one leaf node in case of natural data distribution are then potentially placed in different nodes. This results in more leaf nodes and the need to join more AVBs during query optimization.

We also compared our approach to the work presented in [HHK$^+$10]. For this purpose we ignored the type information and LSBs stored in our synopsis. Despite the partitioning of triples by data source and predicate, this closely resembles the original data summary. In this case the optimizer was not able to handle about 51% of the workload's queries because it produced a huge number of intermediate results during join processing and ran out of memory. Increasing the heap size limit did not effectively solve this problem. The error rate increased with the complexity of the underlying query pattern ranging from 0% for the query class $S_1V_1C_1$ to 98%

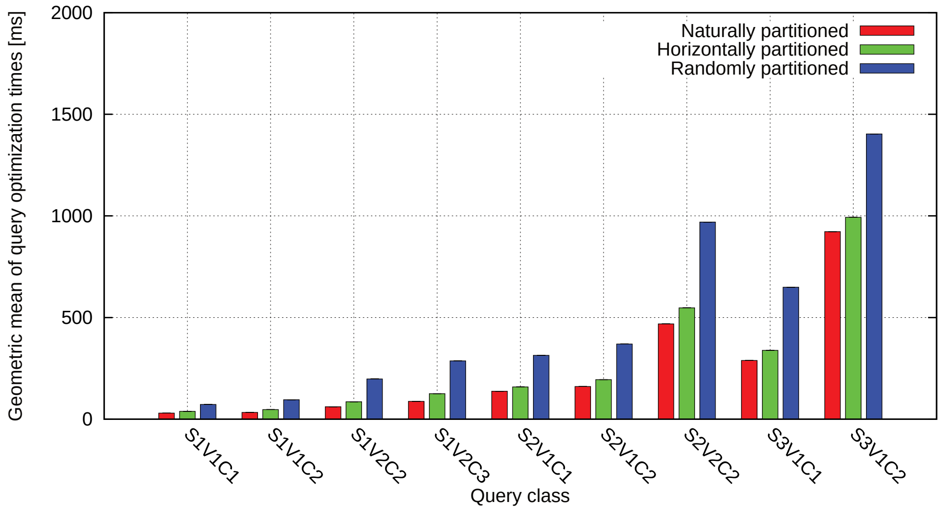


Figure 5.29: Time needed for query optimization

for $S_2V_2C_2$, $S_3V_1C_1$ and $S_3V_1C_2$ respectively. The running times for optimizing the remaining queries increased by an average factor of about 50, ranging from 1.6 for the query class $S_1V_1C_2$ up to 288 for $S_3V_1C_1$.

Query Execution

Figure 5.30 shows the average time needed to execute the workload including query optimization. With a workload-average of about one second in case of natural and horizontal partitioning the prototype performs very well.

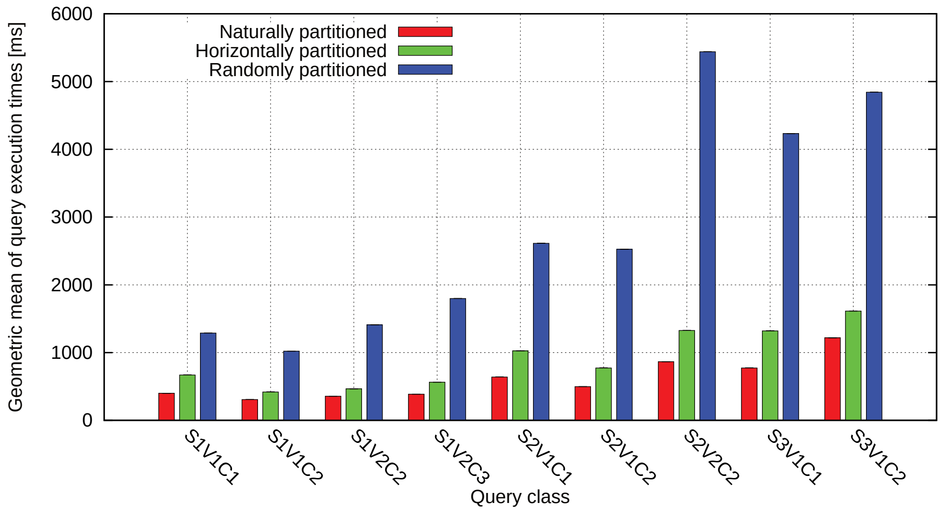


Figure 5.30: Time needed for query execution

The average query execution times increase by a factor of up to five when the data is partitioned randomly. In contrary to the other scenarios, where most triple patterns can be answered by only one single data source, there are often much more potential sources in this scenario. As a result, less triple patterns can be pushed down and grouped into more complex subqueries which significantly increases the number of subqueries and resulting variable bindings. Although many of these bindings can be pruned, this increases local query execution times because the reducers are only applied to the result sets of the subqueries and are not directly involved into local query processing.

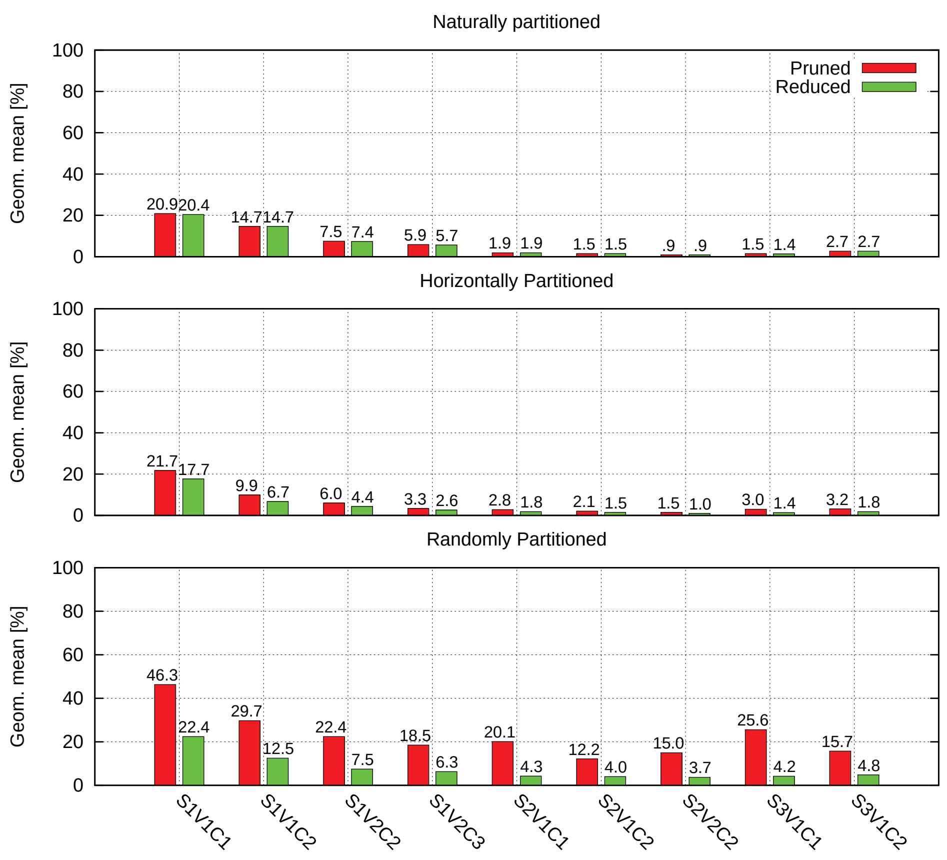


Figure 5.31: Average execution times for the different query classes

The impact of the different optimizations is shown in more detail in Figure 5.31. We denote pruning parts of the operator tree with *Pruned*, and further reducing the number of returned variable bindings with *Reduced*. The baseline of 100% is defined by the unoptimized case, in which the step of query simplification is omitted (see Section 5.3.1) and initial execution plans are post-processed and executed directly without applying any optimizations. In case of natural data distribution, pruning operators reduces the average query execution time by up to two orders of magnitude ($S_2V_2C_2$). Further reduction of the resulting variable bindings yields only an additional speedup of up to 10%. In total, the average query execution time is decreased by a factor of five ($S_1V_1C_1$) to 110 ($S_2V_2C_2$). In this case applying reducers has only little impact, as subqueries do not yield many irrelevant results. When the data is distributed horizontally, plan simplification yields a speedup of a factor of five to 65 ($S_2V_2C_2$). Reducers also have a significant impact, further decreasing query execution times by a factor of up to two. Applying all optimizations the average query execution time is decreased by a factor of five ($S_1V_1C_1$) to 100 ($S_2V_2C_2$). In case of random data distribution much more data sources yield relevant results, limiting the impact of plan simplification. This results in a speedup of a factor of two to eight ($S_2V_1C_2$). On the other hand the potential for pruning irrelevant results is significantly increased, leading to an additional speedup of a factor of up to five ($S_3V_1C_1$). This adds up to a total speedup of a factor of four ($S_1V_1C_1$) to 25 ($S_2V_2C_2$).

When compared to natural data distribution, the overall query execution times increase slightly in case of horizontal and significantly in case of random data distribution. For naturally or horizontally distributed data only a single data source returns

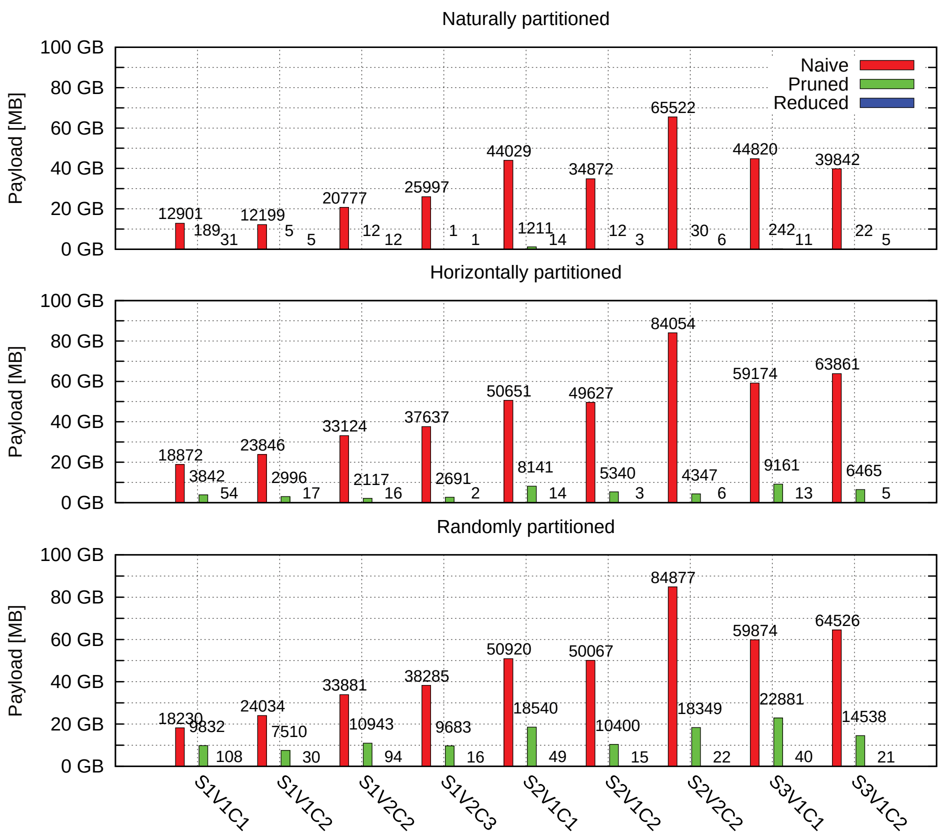


Figure 5.32: Transferred data volumes for the different query classes

relevant bindings for most triple patterns. Therefore these scenarios benefit the most from pruning operators from the query execution plan. Horizontal distribution does also offer some potential for reducing the resulting variable bindings, as the unnatural distribution slightly increases the number of irrelevant variable bindings returned by the subqueries. In case of random partitioning pruning parts of the operator tree is more difficult because relevant results for triple patterns can often be returned by more than one data source. On the other hand this significantly increases the potential to reduce the amount of resulting variable bindings. Independent from data distribution the speedup increases with the complexity of the underlying query pattern.

Details on the total data volume (over all queries in the respective class) transferred while executing the workload are shown in Figure 5.32. Here, *Naïve* denotes the unoptimized case. Plan simplification reduces the transferred data volume by several orders of magnitude. Also pruning irrelevant variable bindings yields an additional reduction by up to another few orders of magnitude, especially in the case of random data distribution. Due to plan simplification the generated subqueries often correspond to the stars contained in a query when data is distributed naturally or horizontally. The number of bindings for these stars are limited to $\leq$10.000 and therefore the number of bindings returned by all subqueries is often $\leq 10.000 * \#Stars$. Loading a dataset into RDF-3X is very fast for smaller datasets. The total query execution times are thus often dominated by local query processing in this cases, because the generated reducers do not affect local query processing. If data is distributed randomly the number of results returned by the subqueries increases significantly. As reducers are able to prune many of these bindings, they do have a strong impact on the overall running times in this scenario.

5.5.2 Evaluation with FedBench

FedBench is the first benchmark that has been proposed for federated SPARQL query processors [SGH+11]. It builds upon a rich collection of publicly available RDF databases, many of which are also part of the evaluation datasets utilized in the previous section. The benchmark consists of four scenarios, a *Life Sciences* scenario, a *Cross Domain* scenario, a *Linked Data* scenario and parts of *SP²Bench*, which is a benchmark for stand-alone triple stores [SHLP09]. The life sciences and cross-domain scenarios include real-world queries over popular RDF datasets. The linked data scenario contains rather simple queries, which are meant for benchmarking Linked Data query processors. The SP^{2}Bench scenario is included for evaluating the completeness of a system's support for all SPARQL features.

Dataset	#Triples	Machine
DBPedia	43.6M	A
NY Times	335k	A
SW Dog Food	104k	A
Geonames	108M	B
Jamendo	1.05M	B
Drugbank	767k	B
Chebi	7.33M	C
LinkedMDB	6.15M	C
KEGG	1.09M	C
Total	168M	–

Domain	Query	Shape	#Results
	1	–	1159
	2	–	333
	3	Hybrid	9054
Life Sciences	4	Hybrid	3
	5	Hybrid	393
	6	Hybrid	28
	7	Hybrid	144
	1	Star	90
	2	Star	1
	3	Hybrid	2
Cross Domain	4	Chain	1
	5	Chain	2
	6	Chain	11
	7	Chain	1

Table 5.3: FedBench datasets and queries

Similar to previous evaluations with FedBench, e.g., [SHH+11] and [GS11], we focus on the life sciences and cross-domain scenarios. In total, the datasets utilized by FedBench contain more than 168 million triples, which results in a synopsis of size 923 MBs that was generated in about 40 minutes. The datasets were distributed amongst the same machines that have been utilized in the previous section. An overview over the individual datasets as well as basic characteristics of the two different query classes is shown in Table 5.3. We did not split the datasets by scenario and, in contrast to other evaluations, also added the dataset from the Linked Data scenario to obtain three datasets per node. The FedBench queries form star-shaped, chain-shaped or hybrid query patterns with up to 9054 results and six joins. The most complex query, query four from the life sciences scenario, forms a hybrid query pattern with six joins, bound predicates, six variables and two constants. As our current prototype does not support *OPTIONAL* and *FILTER* operators, we removed them from the query set. In case of an optional pattern, we executed the query twice, with and without the optional patterns and added up the execution times.

Query Optimization

As specified in [SGH+11], we executed all queries five times following an initial warm-up run. Table 5.5 shows the execution times of our query optimizer for the evaluation queries. It also presents a comparison with the execution times measured when disabling sideways information passing. The numbers clearly show that our optimizer efficiently handles complex real-world queries over large datasets while employing sophisticated multi-pass optimization techniques.

	Reduced			Merged			Bound Predicates			Bound Joins			Mode		
Query	Opt	Total	Reduct	Opt	Total	Reduct	Opt	Total	Reduct	Opt	Total	Reduct	MG	BP	BJ
LS-1	0.01	0.04	0.00	0.01	0.03	0.00	0.01	0.11	0.00	0.01	0.09	0.00			
LS-2	0.70	902.23	99.99	0.70	902.23	99.99	0.68	256.11	99.99	0.69	1.14	2.34		X	X
LS-3	2.90	9.95	4.16	2.90	9.95	4.16	2.38	9.34	4.16	2.37	9.63	4.16			
LS-4	0.16	2.00	99.43	0.16	2.00	99.43	0.15	1.92	99.43	0.16	2.09	99.43			
LS-5	5.41	8.35	96.12	5.41	8.35	96.12	5.91	8.67	96.12	5.50	8.50	96.12			
LS-6	0.04	0.71	99.28	0.04	0.71	99.28	0.03	0.72	99.28	0.03	0.76	99.28			
LS-7	1.34	5.67	94.76	1.34	5.67	94.76	1.15	5.84	94.76	1.20	5.94	94.76			
CD-1	0.14	122.14	99.99	0.18	0.54	0.00	0.14	0.53	0.00	0.15	0.51	0.00	X		
CD-2	0.01	0.26	0.00	0.01	0.26	0.00	0.01	0.25	0.00	0.01	0.25	0.00			
CD-3	0.05	0.61	99.98	0.05	0.61	99.98	0.05	0.59	99.98	0.05	0.67	99.98			
CD-4	0.06	0.75	99.91	0.06	0.75	99.91	0.06	0.76	99.91	0.06	0.76	99.91			
CD-5	0.05	0.58	96.38	0.05	0.58	96.38	0.05	0.59	96.38	0.05	0.58	96.38			
CD-6	0.02	0.35	98.97	0.02	0.35	98.97	0.02	0.34	98.97	0.02	0.33	98.97			
CD-7	0.38	86.02	99.99	0.38	86.02	99.99	0.38	86.16	99.99	0.40	1.37	99.52			X
Total	11.27	1139.66		11.30	1018.05		11.01	371.91		10.71	32.61				

Table 5.4: Execution times [s] and reduction factors [%] for FedBench

In addition, it can be seen that sideways information passing decreases the execution times by up to almost two orders of magnitude for some queries, e.g., query *CD-1*. The execution times include applying all optimizations presented in this paper and building reducers for all subqueries. A more detailed overview of how the different optimizations influence the query optimizer and query execution times is presented in the following section. With the exception of queries *LS-3*, *LS-5* and *LS-7* the optimizer can handle all queries in just a few milliseconds. When the optimization time increases, queries are characterized by a large number of potential sources.

Query Execution

The results of our evaluation of the different query execution techniques described in the previous sections are shown in Table 5.4. Together with the work presented in [SHH$^+$11] and [GS11], our approach is amongst the few systems proposed so far that can handle all life-sciences and cross-domain queries. Moreover, it is the only system that can handle all benchmark queries without executing bind joins.

	Optimization time [s]		
Query	w/o SIP	SIP	Factor
LS-1	0.01	0.01	1.00
LS-2	13.66	0.69	19.79
LS-3	4.88	2.37	2.05
LS-4	0.17	0.16	1.08
LS-5	4.28	5.50	0.78
LS-6	0.05	0.03	1.44
LS-7	2.37	1.20	1.97
CD-1	12.82	0.15	87.67
CD-2	0.00	0.00	1.18
CD-3	0.26	0.05	5.38
CD-4	0.18	0.06	3.03
CD-5	0.15	0.05	3.09
CD-6	0.57	0.02	30.91
CD-7	0.65	0.40	1.61
Total	40.04	10.69	3.74

Table 5.5: Query optimization

Table 5.4 shows the average optimization time, the average total time including execution as well as the percentage of tuples that have been dropped by the reducers. As can be seen, our system achieves good results even when solely relying on the reducers and not implementing any further run-time optimization techniques. The rightmost columns indicate whether query groups were merged (*MG*) or a query was executed with bound predicates (*BP*) or bind joins (*BJ*).

For query *CD-1* two query groups could completely be merged, which resulted in a reduction of the number of subqueries by a factor of two. Moreover, the transformation results in much more selective subqueries, thus reducing the execution time by more than two orders of magnitude. Predicate binding results in a speed-up of almost up to a factor of four for query *LS-2*. This shows that the selectivity of queries with unbound predicates can be efficiently increased with static compile-time optimizations that come at no additional costs.

Queries *LS-2* and *CD-7* also significantly benefit from binding variables at run-time. Although this technique requires additional optimization-phases to be executed for some subqueries, the query optimization time is not increased significantly. Moreover, the query execution time decreases by up to two orders of magnitude.

Even when employing bind joins, the reducers generated by our optimizer at compile-time still lead to a significant reduction of the size of intermediate results, e.g., reducing the number of transferred tuples by 99.52% for query *CD-7*.

Comparison with Previous Work

In this section we compare the performance of our system with FedX [SGH+11]. We chose FedX for two reasons. First, it achieves very good results in the FedBench benchmark. Second, it is the most elaborated example of a system that solely relies on bind joins for query execution. Additionally, we compare the number of subqueries that are required to answer a query with FedX and SPLENDID [GS11]. We chose SPLENDID because it is an example of a state-of-the-art system that utilizes a synopsis (*VOID* descriptions) for data localization. We deployed *FedX* 2.0 over a set of *Sesame* 2.6.9 repositories that provided SPARQL endpoints via *Tomcat* application servers in version 7.0.21. We utilized the same setup as in the previous experiments.

Execution times [s]		
Query	FedX	Own
LS-1	0.04	0.09
LS-2	0.04	1.14
LS-3	2.74	9.63
LS-4	0.01	2.09
LS-5	0.64	8.50
LS-6	55.73	0.76
LS-7	0.62	5.94
CD-1	0.02	0.52
CD-2	0.01	0.25
CD-3	0.07	0.67
CD-4	0.04	0.76
CD-5	0.02	0.58
CD-6	0.26	0.33
CD-7	0.17	1.37
Total	60.76	32.61

	Service Requests			
Query	FedX	SPL$_A$	SPL$_B$	Own
LS-1	1	1	1	1
LS-2	75	26	26	3
LS-3	5860	18	2	2
LS-4	3	11	2	2
LS-5	1191	17	8	3
LS-6	101806	16	8	2
LS-7	873	4	4	4
CD-1	21	26	26	3
CD-2	2	10	2	2
CD-3	69	19	2	2
CD-4	116	20	4	2
CD-5	52	10	2	2
CD-6	552	10	10	2
CD-7	408	13	5	5
Total	111029	201	102	37

Table 5.6: Comparison with FedX and SPLENDID

A comparison of the overall execution times of FedX and our approach is shown in the left part of Table 5.6. It can be seen that – in this setup – FedX outperforms our system for all queries, despite *LS-6*. The presented execution times do not include source selection via SPARQL `ASK`-queries, as the results are cached and the process can be seen as iterative indexing. For some queries, source selection does introduce a significant overhead, though.

For most queries, the heuristics employed in FedX perform very well which results in the execution of a rather small number of highly selective subqueries. In these cases, our system can not outperform FedX as we employ query optimizations that introduce some overhead. Moreover, most queries are also suitable for being evaluated solely by applying bind joins, as the overall selectivity is determined by one or two highly selective subqueries. When these characteristics do not hold for a query, FedX needs to execute a very large number of subqueries, which is the case for queries *LS-3*, *LS-5*, *LS-6* and *CD-6*. Despite for query *LS-6*, the highly parallelized implementation of block-nested-loop joins in FedX still results in a rather good performance as many of these queries can be executed in parallel. For query *LS-6* our system outperforms FedX by a factor of more than 70. Although our system does not generally outperform FedX, it offers much more balanced execution times. The reason is that we mostly rely on the generated reducers for local query execution and employ further run-time optimizations only for highly unselective subqueries.

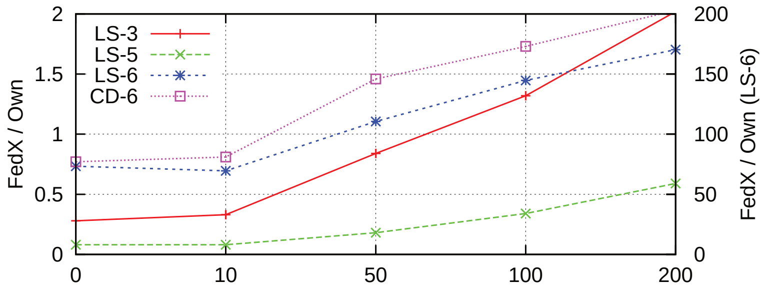


Figure 5.33: Comparison for increasing latency [ms]

Our approach can therefore answer all queries while only executing a very small number of subqueries. This can be seen in the right part of Table 5.6. It shows a comparison with FedX and SPLENDID in terms of the number of service requests, i.e., subqueries. The numbers for SPLENDID are taken from [GS11] and resemble lower bounds for our more complex scenario. SPL_A is a variant of SPLENDID that only utilizes information about predicates contained in a *VOID* description, whereas SPL_B utilizes further schema information and can not handle schema-free datasets. Our approach requires significantly fewer subqueries to evaluate a global query, up to five orders of magnitude compared to FedX and one order of magnitude compared to SPLENDID. The reason for this is twofold. First, our optimizer is able to locate relevant data sources with high accuracy, which minimizes the number of subqueries. Second, these queries return only very few intermediate results due to our reducers. Bind joins can therefore be executed without requiring additional messages to be sent.

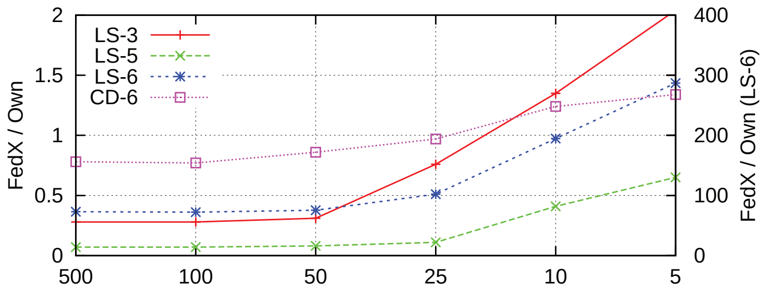


Figure 5.34: Comparison for decreasing maximum concurrent connections per user

The small number of subqueries immediately pays of in resource-constrained scenarios, such as the Semantic Web. Here, it can be assumed that some systems are only reachable via a high-latency network or that administrators limit the number of concurrent connections per user for their published endpoints. Here, the highly parallelized execution of a large number of queries does reach its limits. In Figures 5.33 and 5.34, we show this for *LS-3*, *LS-5* *CD-6* on the left y-axis and *LS-6* on the right y-axis. It can be seen that our approach is much less vulnerable to decreasing resource availability and outperforms FedX for queries *LS-3* and *CD-6* for a network latency of about 75 ms or a maximum of about 10 concurrent connections per user. The difference in execution times for query *LS-6* increases steadily, up to a factor of more than 400. For the other FedBench queries, similar effects can be measured but with a less strongly developed effect. We include *LS-5* as an example of a query were (similar to all other queries) the performance of FedX decreases, but it still outperforms our system under the conditions evaluated in our experiments.

5.6 Summary and Perspectives

In this chapter, a scalable system for optimizing and executing SPARQL queries over large distributed RDF graphs has been presented. Because of the wide-spread use of some RDF vocabularies and the schema-relaxed nature of SPARQL, many RDF databases are potentially able to answer a single triple pattern. But if the same triple pattern is part of a more complex SPARQL query many of these answers are irrelevant due to a lack of join partners. The rich RDF-specific synopsis described in this thesis enables efficient compile-time and run-time techniques that address this problem. At compile-time, sources that would return only irrelevant results for an individual triple pattern can be pruned from the query execution plan. Especially this technique is also of high relevance for querying Linked Data [LIN]. Because this scenario is characterized by a large number of small data sources that are only accessible via a high latency, low bandwidth network, it is very important to minimize the number of subqueries.

In contrast to previous approaches, our system utilizes a portfolio of different techniques depending on query characteristics. We have shown that our comprehensive synopsis allows for highly accurate data localization and enables novel optimization techniques that further reduce the number of subqueries. We have also shown how a query optimizer based on comprehensive summarizations of distributed RDF databases can be implemented efficiently. For query execution, our system employs cardinality estimates to decide at compile-time which execution and optimization techniques should be applied at run-time. This includes efficiently computing distributed joins via compile-time generated reducers or via bind joins executed at run-time. For reducing the selectivity of queries with unbound predicates, information from the global data summary can be utilized to pre-compute a set of relevant bindings. While the presented approach does not generally supersede previous work, we have shown that our system provides more balanced execution times and is less volatile to resource limitations as can be expected in real-world scenarios such as the Semantic Web.

There are several possible directions for future research. For example, our synopsis implements vertical partitioning [AMMH07]. It has been shown that this storage scheme does not perform very well for datasets with a large number of distinct RDF predicates when implemented on top of relational database systems [SGK$^+$08]. Similar problems arise in our solution for very heterogeneous datasets in which a large number of predicates occur only rarely. As this leads to many small partitions the compression ratio can drop significantly which also has a negative impact on running times. This problem can be overcome by merging small partitions into one common partition. A related approach has been proposed in [NM11] for other schema-level information. To improve the performance and results of join-order optimization the concept presented in [NM11] could be adopted. The basic idea is to provide very accurate cardinality estimations based on a lightweight RDF-specific approach for mining parts of a dataset's schema. This could potentially be implemented in our approach based on the triples' subject, predicate and object types and hash values. While our approach also allows to accurately estimate cardinalities after query simplification, characteristic sets could also help to determine the original cardinality of subqueries without the need for additional optimization phases.

In its current state, our optimizer utilizes informed heuristics, based on cardinality estimates, to dynamically apply different strategies for query execution. In future work, we aim at investigating how the comprehensive synopsis can be utilized for implementing cost-based decision processes, e.g., by estimating selectivities for joins between subqueries based on the approximations computed by our optimizer. In order to make effective use of this optimization, the local databases need to be able to process queries containing initial variable bindings. Although there are workarounds, this is currently not supported in most RDF stores but has been included in the W3C's working draft for the upcoming SPARQL version [SPA].

Future work could also investigate how distributed RDF stores can be queried efficiently by processing the results of subqueries with MapReduce. A related system has been presented in [HAR11]. Although the system presented therein also offers query processing over distributed instances of the RDF-3X system, its aim is to use a cluster of machines in order to manage one large RDF dataset. As it needs to control how data is distributed among the local databases, its concepts are not directly applicable to our requirements though.

The prototype presented in this thesis does not require the distributed querying engine to support concurrent reads and writes. In contrast, the summaries are rebuild from scratch at certain points in time in order to incorporate new knowledge into the dataspace (see Chapter 6 and Chapter 7). It is generally possible to propagate updates to our synopsis but the subject, predicate and object hash values and types of triples that have been added, updated or deleted have to be made available. RDF-3X offers a natural way to implement this, as it handles updates via a so-called differential index which is periodically merged with the main index. Our structure will degrade over time when updates are performed because the MBBs of a PARTree's leaf nodes can not be split in a reasonable way without knowing the original data items. Future work could investigate means to measure this degradation for automatically rebuilding the synopsis when required. While offering comprehensive approximations of distributed RDF datasets, our synopsis is well-suited for a read-mostly scenario in which updates occur only rarely. A read-mostly scenario is a common assumption made by all state-of-the-art distributed SPARQL query processors for source indexing (e.g., [SVHB04, QL08, BB10]) or caching (e.g., [GS11, SHH+11]).

CHAPTER 6

Bridging the Gaps: Semantic Integration and Data Transformation

With the techniques presented in the previous sections it is possible to provide an integrated access to a set of distributed RDF representations of biomedical data sources. Conceptually, the most important missing components of a comprehensive information integration platform are flexible and scalable techniques for semantically integrating and transforming data. This chapter first presents approaches for manually editing and annotating the dataspace for semantic integration purposes. This includes techniques for browsing the dataspace, which is challenging as it consists of a large distributed graph structure. Secondly, a flexible technique for automated semantic integration is presented, which is able to leverage the manually defined annotations. It is able to access the querying interface presented in the previous section and to post-process the result sets by executing transformation workflows, which are formulated in a scripting language. A real-world scenario includes a large number of such transformation scripts. The system is therefore able to automatically parallelize the execution of the individual scripts based upon knowledge about the dependencies between them. It further automatically distributes their execution over the available computing nodes for better resource utilization. Finally, we describe different domain-specific operators, which can be utilized by the integration scripts. This operators, e.g., allow to execute semantic reasoning or data de-identification processes.

6.1 Manual Annotation and Mapping

When annotating and editing a graph-structure, it is important to provide a suitable overview over the data. Therefore, this section focuses on browsing and navigating large RDF graphs. At any time during this process, the approach allows data integrators to edit existing data, or add new data. In the latter case, the user is required to specify the target location (database) for the new data. It can be assumed that the user is able to utilize multiple navigation- and editing-components in parallel in order to compare the data and map data items to each other. The system discriminates between *low-volume data* and *high-volume data*, which might be too large to be displayed as a whole or does not fit entirely into main memory.

6.1.1 Related Work

In case of high-volume data, a component is needed which allows to browse the graph, as it is not possible to display all information at the same time. In [DK08] an approach has been presented, which is based upon a layered representation of the graph. It does not consider the direction of the edges in the graph, but builds a tree-structure with a circular layout following the direction of the navigation. When a new node is added, it is put one layer below its parent node and shares the space induced by the angle of its parent with its siblings. If this angle is not sufficient to provide space for all of a node's children, it calls its parent and requests more space, which can be achieved by recomputing the angles. This request for more space is passed on recursively until it can be fulfilled. If the request reaches the root node, all nodes are moved one layer away from the root node to provide more space. The approach also includes concepts for deleting and reorganizing nodes. A drawback is that it displays the graph as a tree-structure, although this is not suitable for many graphs. For example, if a graph is strongly connected, there are many edges which are not part of the navigation tree and thus complicated to navigate and layout. Furthermore, the resulting layout is dependent on the order in which the graph is navigated and can not be easily reproduced. The approach presented in [ODD06] implements a different concept. Here, the data is not displayed as a graph, but in tabular form. Querying is implemented via so-called "facets". A facet normally represents a property or classification of the nodes. In case of patient-data this could, e.g., be the name or the date of birth. In a graph, facets would be the incoming and outgoing edges of a node. The graph is navigated by adding, removing and altering facets. Moreover, it is possible to query the dataset by defining restrictions for facets. The concept is oriented towards semi-structured data, which follows a meaningful schema, and is thus not suitable for our application context.

Smaller datasets can be annotated and altered by utilizing an editor for RDF data, which loads and displays the entire dataset at the same time. Protégé is an open-source platform, which includes multiple tools for creating, editing and managing ontologies [GMF$^+$03]. It offers different ways to create knowledge bases, supports many formats and is in wide-spread use. It is not suitable in our context, because it implements a schema-first approach. This means that most of its functionality (e.g., a graphical view on the data) is only available after the ontology or schema of the data has been specified. In [TOP] a commercial workbench for creating and editing ontologies is provided. Similar to Protégé, it is strongly oriented towards the meta-level, i.e., it implements a schema-first approach. It provides extensive means to graphically navigate and edit ontologies. [REJ] is a very light-weight open-source editor for RDF data. It is well suited for the rapid development of small ontologies and is based upon a textual representation. In contrast to [GMF$^+$03] and [TOP] it does not enforce a schema-first approach and is easy to use. It is not suited for our context, because it is oriented towards very small datasets.

6.1.2 Navigating the Dataspace[1]

Within the context of a large distributed RDF graph, performance aspects are highly relevant. This includes techniques to reduce the main memory consumption and to achieve an acceptable response time by implementing a suitable caching strategy to reduce the number of queries. This section presents the basic concepts behind our approach, whereas details on the resulting user interface are given in Chapter 7. The

[1]Parts of the work presented in this section are based on the student project [Kuh11]

basic idea is to allow the user to explore the distributed graph structure by expanding and collapsing nodes, i.e., displaying or hiding nodes and its children or ancestors. The resulting sub-graph is displayed by applying a spring layout algorithm, which dynamically adjusts to the graph structure. Two aspects of this approach have to be implemented carefully. Firstly, it must be ensured that the displayed sub-graph is not too large, as this reduces visual clarity and consumes a lot of resources. Secondly, it must be ensured that this also holds after a node has been expanded. The reason is that in a strongly connected graph, a node might have a large number of children or ancestors, which can not be displayed at the same time when it is expanded. The component implements different designs for resources as well as literals and utilizes prefixes to shorten the resources' identifiers. This prefixes are managed in a consistent way for all components in the overall integration system and are, e.g., coupled to the common prefixes which are determined during indexing (see Section 5.2). For performance reasons, the system maintains a global in-memory cache, which implements a Least Recently Used (LRU) eviction policy, and maps URIs to their abbreviated representations. For navigating the RDF graph, the component solely relies on the querying interface, which has been presented in the previous section. The incoming and outgoing edges of a given node n can be retrieved by executing the following SPARQL queries:

- Incoming: `select distinct ?s ?p where ?s ?p` *n* `order by ?p ?s`.
- Outgoing: `select distinct ?p ?o where` *n* `?p ?o order by ?p ?o`.

Expanding Nodes

The expansion of a node is a difficult operation, as any node might have a large number of incoming or outgoing edges. For example, a resource which defines a class that is frequently used in a dataset might be strongly connected. It is therefore necessary to, firstly, provide means to display only some of those edges and to allow the user to select which ones to display. Secondly, it is important to cache the neighbourhood of a displayed node and provide means to browse and search it. The component initially only displays n edges for each node, whereas n is a user-defined parameter. The other edges can be accessed by browsing through a list and added to or removed from the sub-graph. As this list might be too large to fit into main memory, it is stored on disk where it is organized into pages. This allows for a fast access to all elements within a certain page, which are read dynamically from the disk during browsing. In addition, the lists are also used as a local cache. Although, it would also be possible to use a database for this purpose (e.g., an RDF database), databases generally do not provide an interface for accessing the data organized in a manner suitable for this process.

An example of the implemented index structure is shown in Figure 6.1. The *Nodes index* is a hash map, which allows to retrieve the list of all pages (*Page index*) for a given node. Those pages are stored on disk and can be retrieved dynamically. A LRU eviction policy ensures that the in-memory index does not grow too large and also frees the memory on the hard-disk if a node is evicted from the index. Due to this design, the component has low memory footprint but allows to efficiently browse a large strongly connected RDF graph.

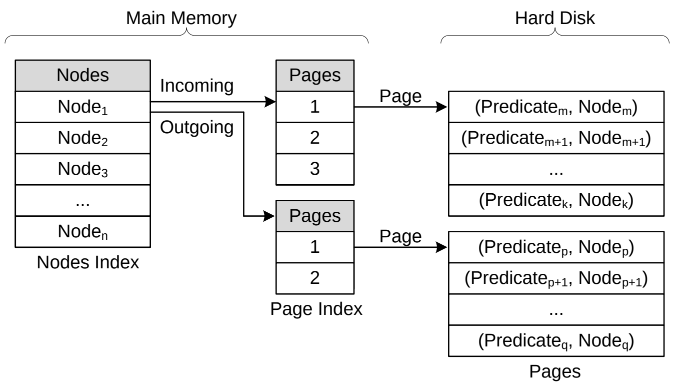


Figure 6.1: Index structure for navigating an RDF graph

Collapsing Nodes and Cleaning Up

For visual clarity and lower resource consumption the size of the navigated sub-graph is adjusted dynamically during browsing. To this end, all visible nodes are added to a priority queue based upon a time-stamp of the last interaction (expanded, edges added etc.). If the total number of nodes exceeds a predefined limit, this priority queue is utilized to remove the least recently used nodes from the sub-graph until the limitation holds again. Removing a node means to remove the node itself as well as all of its incoming and outgoing edges. After removing a node, a clean-up operation is performed which removes orphaned nodes. To this end, all nodes which are not connected to any other node any more are also removed from the graph. Nodes are removed in the same way, when they are explicitly collapsed by the user.

6.1.3 Editing Low-Volume Data[2]

In case of smaller datasets, it is possible to display the complete graph for editing and annotation, which allows to include an easy-to-use RDF editor into the system. The provided RDF editor does not enforce a schema-first design, but is basically a graph editor in which node labels are resources or literals and edge labels are resources. The editor is fully integrated with the overall prefix management and performs simple consistency checks, such as making sure that subjects are always resources and that URIs and literals have a valid syntax. The approach is well suited for editing medium size datasets similar to [GMF+03] and [TOP] but is as flexible as [REJ].

6.2 Automated Data Transformation [3]

After a dataset has been made accessible to the integration system and potential annotation and mapping steps have been performed, complex transformations can be applied to the data. To this end, a workflow engine is presented in this section. It allows to formulate transformation and integration processes in a scripting language, which is able to access the distributed data sources via the querying interface. When executed, each script can issue queries to the dataspace and relate as well as post-process the returned result sets. As the underlying system is inherently distributed, two major optimizations are presented. This includes an algorithm for automatically deriving a plan, which tries to maximize the level of parallelism when executing the

[2]Parts of the work presented in this section are based on the student project [Kuc11]

[3]Parts of the work presented in this section are based on the student projects [Fre12], [Zie12] and [Web12]

workflow. Furthermore, a load-balancing component distributes the execution of the scripts to the available computing nodes in order to leverage all resources.

6.2.1 Related Work

Transforming data and materializing implicit information are common tasks in data management, e.g., [SDB10]. In the context of the Semantic Web, the most important approaches include semantic reasoning techniques and rule-based systems for transforming RDF data. The *Rule Interchange Format* (RIF) is a W3C recommendation for exchanging rules in the Semantic Web [RIF]. In this context, rules are used to formalize knowledge and RIF tries to bridge the gaps between the many dialects available. It is closely tied to OWL and RDF. Simple statements of the form `IF <CONDITION> THEN <EXPRESSION>` form the core dialect of RIF. This is further extended by a *Basic Logic Dialect* and a *Production Rule Dialect*. In general, rule-based systems are based on simplified first-order-logic, although any rule system can implement its own syntax and semantics, such as RIF SILK [SIL].

In the context of this thesis, rule-based systems act as a stimulus for the development of a more specialized approach, which needs to handle aspects of data access (i.e., creating, reading and writing databases) and tries to exploit the fact that the overall system architecture is inherently distributed. The development of comprehensive semantic reasoning techniques has been started by early work in the area of deductive database systems [RU93]. Different semantic reasoning approaches for RDF/S and OWL ontologies have been presented in [JEN, HER, PEL, MS05]. These components can be integrated into the engine presented in this thesis and will be explained in more detail in Section 6.3. The main contribution of this section is a workflow engine which is able to execute data transformation scripts, each of which can utilize external components which implement data de-identification, semantic reasoning or arbitrary data transformations. Analogously to many languages for rule-based systems for the Semantic Web, our data transformation scripts access the underlying dataspace via SPARQL queries and construct new knowledge by applying further operators. A similar approach for ontology-alignment has been proposed in [EPS08].

```
READS = { DB_A, DB_B }
WRITES = ∅
CREATES = { DB_C }

\* Create a new database *\
Database DB_C = IO.createDatabase("DB_C", "Site_1", PERSISTENT);

\* Materialize implicit knowledge *\
TF.materialize("DB_A", "DB_C");

\* Add data from DB_B *\
IO.add("DB_B","DB_C");
```

Figure 6.2: Example data transformation script[1]

[1]Here and in the remaining examples, the definition of databases and sites has been simplified for better readability. In the preamble (READS, WRITES, CREATES), databases are defined by a unique identifier. This definition is supported by the graphical user interface presented in Chapter 7. Moreover, the scripting environment offers interfaces to a registry for accessing databases and sites via labels or identifiers.

The individual scripts are executed by a dedicated workflow engine. This approach has also been taken in many other projects, e.g., [GNTT10, OLHP+10, TAJ+10]. Similar problems arise in the area of workload management, where the aim is to manage dynamic database workloads for meeting complex service level objectives [KSA+08]. Our approach is different from typical workload managers, as it needs to consider complex dependencies between the managed entities and is not able to leverage any in-depth information, e.g., to build a cost model. Various optimization techniques for related scenarios have been developed in the context of research on data stream management or complex event processing, e.g., [ABB+04]. Here, the aim is to efficiently process streaming data, e.g., by sharing data streams between operators [KSKR05]. Our problem is different from complex event processing, as our operators are not able to process data in a pipelined manner. For instance, semantic reasoning or data anonymization processes always require a global view on a materialized dataset.

6.2.2 Basic Concepts

The functional units managed by the workflow engine provide means to define integration and transformation processes in an expressive scripting language. To this end, we utilize JavaScript in our prototype, although it would of course be possible to develop a dedicated *Domain Specific Language* (DSL) tailored towards the application scenario. Within these scripts, two interfaces provide important functionality to read and write data (*IO*) and to transform data (*TF*). On a meta-level, each script needs to define the datasets that it reads, writes and creates. This information is later utilized for implementing several optimizations. An example is shown in Figure 6.2.

In this script a new database DB_C is created which contains all information from the database DB_B as well as all implicit and explicit knowledge from the database DB_A. This means that an OWL reasoner has been executed, which materialized all implicit knowledge within the dataset. The interfaces *IO* and *TF* provide a scripting environment which consists of several methods for accessing and transforming the datasets. An overview over the most important operators for reading and writing the dataspace are shown in Table 6.1.

Method	Description
`Query(Query)`	Executes a global query and returns a result set
`Query(`DB_1`,...,`DB_n`, Query)`	Executes the query on the databases and returns a result set
`CreateDatabase(Site, Name, Type)`	Creates a new global database
`CreateDatabase(Name, Type)`	Creates a new local database
`Add(`DB_1`, `DB_2`)`	Adds the data from DB_2 to DB_1
`Add(DB, `Triple_1`,...,`Triple_n`)`	Adds the triples to the database
`Construct(DB, ResultSet, Pattern)`	Constructs triples and adds them to the database

Table 6.1: Overview over the `IO` interface provided for scripting

As can be seen the *IO* interface offers several methods for creating databases (`CreateDatabase(...)`), querying an arbitrary subset of the available databases (`Query(...)`), adding data to a database (`Add(...)`) or constructing new data out of a query's result set (`Construct(...)`). The table only shows the most general specifications of each function, although more specialized variants are provided by the interface. The construct method is similar to the construct-variant of SPARQL queries [PS08] and is used in the example script in Figure 6.17. The methods for creating new databases have a parameter `Type` and allow to create four different types of databases:

- `CreateDatabase(Name, VOLATILE)` creates a temporary local in-memory database. This means that the database is kept in main-memory and will be dropped after the script has terminated.
- `CreateDatabase(Name, PERSISTENT)` creates a temporary local database. This means that the database is stored on disk and will be dropped after the script has terminated.
- `CreateDatabase(Site, Name, VOLATILE)` creates a temporary global database. This means that it is stored on disk at the specified site and will not be dropped until all scripts have terminated. The database is not indexed by the mediator and can thus only be queried by other scripts as a single non-integrated database.
- `CreateDatabase(Site, Name, PERSISTENT)` creates a persistent global database. This means that the database is stored on disk at the specified site and will never be dropped. Additionally, the database is indexed by the mediator and can be queried in an integrated manner by all other scripts in the workflow.

From the perspective of a workflow engine, the first three ways in which databases can be created are easy to handle. Creating a global persistent database is more difficult to implement, however, because the database has to be available to other data transformation scripts for integrated querying. To this end, the database is indexed temporarily after the creating script has terminated. Temporary indexing means that the global prefix tree (see Section 5.2) will not be built in the balanced manner described in Section 5.2. Instead the global prefix tree is built incrementally by merging it with the local prefix trees. If the overall number of prefixes exceeds the limit of about 2^{16}, the tree is not re-balanced to include the top-k prefixes, but instead the overflowing prefixes will be assigned to the unknown category. In the rare cases where the number of predicates did exceed the limit while executing the workflow, the global synopsis (including the prefix tree) will be rebuilt from scratch after the termination (see Chapter 7 for more details). The same mechanism applies for databases which are altered during the execution of the data transformation phase.

Regarding data transformation, robustness and the principle of loose coupling (see Section 3.2) are implemented at the level of data transformation scripts. To this end, the workflow engine guarantees atomicity by ensuring that the results of a script are either persisted completely or not at all if the script is subject to any errors. This is implemented by rolling-back all changes created by a script in case of any error. The workflow engine will always execute all available scripts regardless of any local errors. On the application level (see Chapter 7), data integrators are informed visually about any errors, which allows to adjust the scripts in order to fix the problems.

Method	Description
`Materialize(`DB_{in}`, `DB_{out}`)`	Materializes implicit knowledge
`Pseudonymize(`DB_{in}`, Query,` DB_{idat}`,` DB_{pseudo}`,` DB_{mdat}`)`	Pseudonymizes the database DB_{in}
`Anonymize(ResultSet, K, Suppression, Metric, Hierarchy`$_1$`,...,` `Hierarchy`$_n$`)`	Creates a k-anonymized version of the result set
`GetHierarchy(Type, Name)`	Creates a generalization hierarchy

Table 6.2: Overview over the `TF` interface provided for scripting

An overview over the most important methods provided for transforming datasets (*TF* interface) is shown in Table 6.2. The method `Materialize(...)` provides the most important functionality for semantic integration and executes an RDF/S and OWL reasoner, which materializes all implicit A-Box (instance-level) and T-Box (schema-level) statements in the specified database. It is explained in more detail in Section 6.3. The function `Pseudonymize(...)` creates a pseudonymized version of the input database. As input it requires a query, which selects all identifying data and includes variable named *id* which denotes the resources that identify patients. The result is written to three datasets. The dataset DB_{idat} will receive a copy of the data selected via the query, the dataset DB_{mdat} will receive the remaining data in which the patient identifiers have been substituted with pseudonyms, and the dataset DB_{pseudo} will receive a mapping between the original identifiers and the pseudonyms. This mechanism allows to separate different types of data (e.g., identifying and medical data) and to grant different groups of users access to different types of data without leaking any of the original identifiers. The method `Anonymize(...)` provides means to create a de-identified version of a given result set and will be explain in more detail in Section 6.4. The implemented anonymization algorithm is based upon generalization hierarchies, which can be generated automatically for some common attributes by calling the function `GetHierarchy(...)`. Although the scripting environment only provides a very small set of methods for data transformation, it offers a powerful mechanism to implement complex semantic integration tasks. This is due to the integration of a global view on all datasets, which enables data integrators to issue complex queries against the dataspace. In combination with user-provided annotations and an expressive scripting language, this allows to automatically evaluate complex expressions in order to generate new knowledge. The scripting environment can be extended by integrating many of the available tools for processing RDF data, such as rules engines or entity resolution techniques (see Section 6.5).

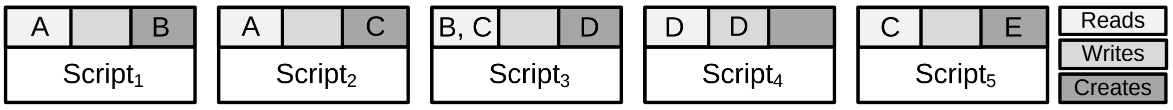


Figure 6.3: Example set of transformation scripts

For the workflow engine, each script is a black box with associated data access characteristics. An example scenario is shown in Figure 6.3. It consists of five integration scripts, which operate on five different databases (A-E). These scripts are executed by a workflow engine which tries to maximize the level of parallelization and to achieve a balanced load distribution. These two optimizations are presented in the following section. They are based on the information about which database is read, written or created by which script as well as optional user-feedback and some annotations.

6.2.3 Optimizations

A realistic scenario consists of a large number of integration scripts and a large number of distributed computing nodes. We have therefore developed an approach which aims at leveraging the available resources as much as possible during data transformation. To this end, it first computes a parallelized execution plan containing all predefined integration scripts. This execution plan is built by analyzing the data dependencies between the scripts and is combined with an optional user-feedback phase. An overview over this process is shown in Figure 6.4.

During the execution phase, the parallelized plan is executed on a single node in a

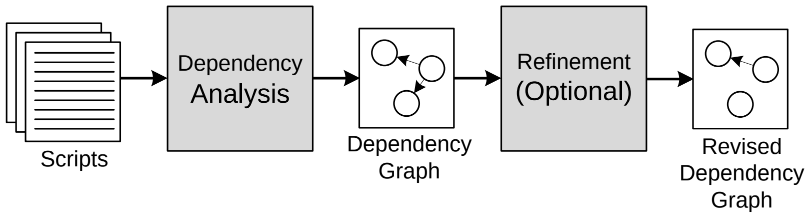


Figure 6.4: Optimization of data transformation scripts - Step 1

multithreaded manner. Whenever a transformation script is to be executed, the parent thread calls a scheduler component and waits until the script has been executed. The scheduler distributes the plans amongst the computing node in order to balance the load. This dynamic approach is required, because the the workflow engine does only have very little knowledge about the characteristics of the integration scripts in terms of resource consumption. The reason for this is, that the scripts are user-defined and may implement arbitrary transformation processes. In general, it is also not possible to monitor the scripts' behaviour and utilize this information for subsequent executions of the workflow, because the scripts access the underlying datasets, which might change significantly over time. Finally, a dynamic approach is also important because the available computing nodes might execute other tasks, such as queries issued by users (see Section 7.1.3). Although implementing a dynamic approach, the scheduler incorporates some static information about the workload, e.g., for prioritization purposes. An overview over this process is shown in Figure 6.5.

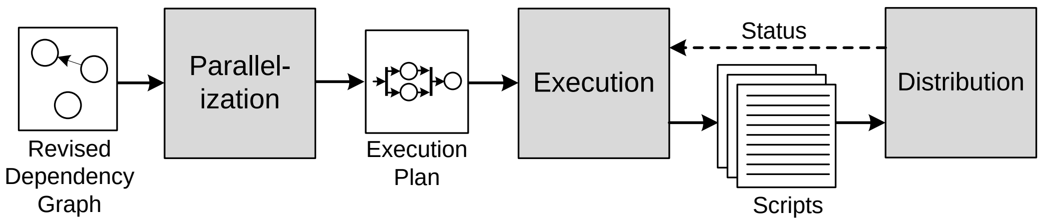


Figure 6.5: Optimization of data transformation scripts - Step 2

Parallelization

In a real-world scenario, a large number of integration scripts would be defined. Although there are some dependencies between the scripts, many can be executed in parallel while still leading to a consistent and correct result. To this end, we have developed an optimization component which is able to automatically derive a parallelized execution plan for a set of integration scripts. These execution plans are computed based upon known dependencies between the data flow characteristics of the scripts. The dependencies are derived from the defined input- and output-databases and are shown in Figure 6.6 for the example scripts. Here, the scripts 3 and 5 depend on script 2, script 3 depends on script 1 and script 4 depends on script 3. A script depends on another script if it queries a database, which is written or created by the other one. Script 4 depends on itself, as it reads and writes to the same dataset. Circular dependencies between the scripts can not be handled by the workflow engine. For example, it would need to be guaranteed that every script reaches a fixpoint when executed repeatedly. To this end, circular dependencies are removed from the dependency graph. This can be done automatically by proposing candidate dependencies for deletion, or during the user-feedback phase. Afterwards, topological sorting is utilized to derive a parallelized execution plan [CSRL01].

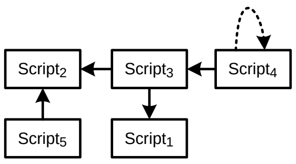


Figure 6.6: Dependencies between the example transformation scripts

The (optional) user-feedback phase allows data integrators to manually alter the dependency graph in order to reach a higher level or parallelism. For example, a script might write to a database which is read by another script but both scripts touch distinct subsets of the contained data. The user is provided with a graphical view of the dependencies, which allows to remove redundant edges from the graph. It is important to note that a user has to do this in a cautious way, because removing a dependency which actually exists yields wrong results when executing the plan.

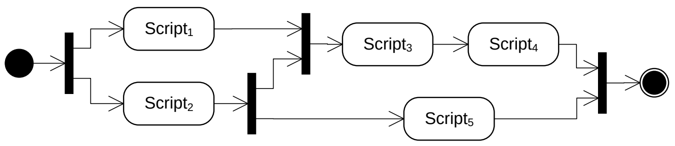


Figure 6.7: Example execution plan as an UML activity diagram

The execution plan for the example is shown in Figure 6.7. The system first needs to execute the scripts 1 and 2, which can be done in parallel. As soon as script 2 has terminated, script 5 can be executed. Script 3 and finally script 4 depend on the termination of the scripts 1 and 2. The circular dependency of script 4 on itself has been removed prior to the plan generation. The process halts when the scripts 4 and 5 have terminated.

Scheduling

The overall architecture of the presented integration solution is inherently distributed. Because many of the involved data transformation scripts could potentially implement algorithms of high complexity and demand lots of resources, it is important to distribute the load amongst all computation nodes. The ultimate goal is to leverage the available resources to execute the plan as fast as possible. In our context, a computing node is a site and each site can host an arbitrary number of datasets. The scheduling algorithm is executed by the *distribution* component from in Figurer 6.5, which is called dynamically by the component which executes the parallelized execution plan. When executing a script, each thread calls the scheduler, which dispatches the execution of the script to one of the available computing nodes. The overall plan which is executed during a data transformation phase, also contains further steps which are automatically and transparently inserted by the integration system. This includes scripts for creating an up-to-date RDF snapshot of all exported data sources and scripts for (re-)indexing each data source before it is accessed by any other script. The scheduler builds upon static information about the integration scripts and dynamically adjusted information about the current state of all computing nodes.

We make some simplifying assumptions, which are valid for our prototype and our testbed. We assume that the distributed system of computing nodes is homogeneous, i.e., all nodes offer the same resources. Moreover, we assume that the network connection offers the same bandwidth between all nodes. These limitations can easily be overcome by implementing minor extensions to the presented approach. An overview

Scope	Parameter	Type	Data type	Description
Node	Datasets	Static	Set of DBs	Datasets hosted by the node
Node	Cores	Static	Integer	Number of CPU cores
Node	RAM	Static	Integer	Overall main memory
Script	Reads	Static	Set of DBs	Datasets read by the script
Script	Writes	Static	Set of DBs	Datasets written by the script
Script	Creates	Static	Set of DBs	Datasets created by the script
Script	Priority	Static	Integer	Priority of the script
Script	MPL	Static	Integer	The number of concurred threads
Script	RAM	Static	Integer	The main memory requirements
Script	IO-bound	Static	Boolean	Flag for disk-IO-bound processes

Table 6.3: Static and dynamic parameters for load distribution

over the parameters utilized by our scheduler is shown in Table 6.3. For computing nodes, static information includes a shallow specification of each node as well as the hosted datasets. For the individual scripts, the scheduler utilizes information about the input and output behaviour in terms of the read, written and created datasets. It also assigns a *priority* to each script, which is defined as the number of other scripts which depend on the results of its execution. Each script is additionally annotated with information about the number of concurrent threads (*MPL*), the required amount of main memory (*RAM*) and a flag, which determines whether it is a *disk-IO-bound* process, i.e., its load profile is dominated by excessive use of a node's hard disk. Although the specification of this information seems to be a rather time-consuming and error-prone requirement to data integrators, most of these information comes naturally and default values will be adequate most of the time. The default MPL is one, which is correct for all standard scripts which utilize the provided interfaces and also for all automatically generated data export and indexing tasks. In our prototype the required amount of main memory needs to be specified anyway, because each script is executed within its own JVM which needs to define its maximum heap size. Realistic parameters for all standard data transformations can be derived from the evaluations in Section 6.3.2 and Section 6.4.4. Moreover, all standard data transformations are not IO-bound. Although they might sporadically create temporary databases or store temporary results on the hard disk, they perform most tasks in main memory. The automatically generated scripts which perform data extraction and data indexing are IO-bound, though, as their load profiles are dominated by on-disk merge-sorts.

For the definition of our scheduling algorithm, we assume that the execution environment consists of a set of nodes $N = \{n_0, n_1, ..., n_i\}$ and the workload consists of a set of scripts $S = \{s_0, s_1, ..., s_j\}$. We assume that the properties from Table 6.3 are available for each node and script via functions. For example, $cores(n_0)$ returns the number of CPU cores of the node n_0, and $creates(s_0)$ returns the set of databases created by the script s_0. The scheduler processes scripts according to the order induced by $priority(s)$. When choosing a potential target node for their execution, the nodes are prioritized according to how much of the required data is available locally. As the scripts are black boxes, the scheduler only knows which datasets are accessed but not with which selectivity. The priority for assigning a script s to a node n is therefore defined as the number of datasets hosted by n which are accessed by s, i.e., $priority(n, s) = |datasets(n) \cap \{reads(s) \cup writes(s) \cup creates(s)\}|$. For each node n we further maintain a set $wload(n)$ which contains all scripts that are currently being executed on this node. When dispatching a script s to a node n we add it to this set, i.e., $wload(n) = wload(n) \cup \{s\}$. Consequently, when a script s has terminated, we remove it from the set, i.e., $wload(n) = wload(n) \setminus \{s\}$.

Based on this prerequisites we can determine whether the scheduler can dispatch a script to a node without overloading it. It is desirable to prevent overloading a node, because parallely executing scripts while exceeding a systems' resources does not yield any additional speedup. In contrast, it might even have a negative effect on the overall performance, e.g., when the system starts trashing due to exhaustive memory consumption. In this context, it is important to note that the overall integration system is not paused during the execution of a data transformation workflow. As a result, all nodes might be executing additional computing tasks, e.g., answering queries issued by users. However, the aim of not exhausting a node's resources is not to prevent affecting these additional tasks, as the workflow engine can be separated from these process by prioritization on an operating system level. In the Linux-world this can, e.g., be achieved with programs such as `nice` (CPU), `ionice` (Disk-IO) and bandwidth shapers such as `trickle` (Network-IO). In terms of the scripts' memory consumption we implement a different approach by making sure that the workflow engine does not consume all local main memory, but leaves a predefined amount max_m for processing other tasks. The reason is that we want to prevent expensive swapping to occur when a higher prioritized task (such as a user query) is to be executed on a busy machine.

Algorithm 6: ScheduleWindow(window)

```
Input: Window $window = (s_i \mid 0 \leq i \leq k)$
begin
    sort $window$ descending by $priority(s_i)$
    foreach $s \in window$ do
        $nodes \leftarrow (n \in N \mid available(n, s))$
        sort $nodes = (n_j \mid 0 \leq j \leq l)$ descending by $priority(n_j, s)$
        if $nodes$ is not empty then
            dispatch $s$ to $n_0$
        else if $window$ is $primary$ then
            $secondary = secondary \cup \{s\}$
```

The scheduler also aims at preventing non-effective parallelism. This applies to executing a set of disk-IO-bound processes in parallel. In this case, as long as the processes perform mainly sequential IO (which is true for all standard scripts in our prototype), executing them in parallel will at best yield the same results as executing them sequentially. Note that, if a computing node would provide multiple disks that are not combined in a RAID configuration, scripts could utilize those independent disks in parallel. This is, however, a rather uncommon configuration, which results in additional optimization problems. Non-effective parallelism is also relevant for CPU resources, as exceeding the number of available physical CPU cores will also not result in any speedup. In such situations, degrading to (locally) sequential execution will allow the scheduler to prioritize the execution of the workload.

The maximum amount of main memory that will be consumed by the workload currently executed on a node n is:

$$\mathbf{ram}(wload(n)) = \sum_{s \in wload(n)} ram(s)$$

Analogously, the current workload of a node n will not utilize more CPU cores than:

$$\mathbf{mpl}(\mathit{wload}(n)) = \sum_{s \in \mathit{wload}(n)} \mathit{mpl}(s)$$

Finally, the current workload of a node n is io-bound if:

$$\textbf{io-bound}(\mathit{wload}(n)) = \bigvee_{s \in \mathit{wload}(n)} \textit{io-bound}(s)$$

Taking all these aspects into consideration, the function which defines whether a node n has enough resources to process a given script s is defined as follows:

$$\mathit{available}(n, s) = \begin{cases} \mathit{false} & \textbf{if } \textit{io-bound}(s) \wedge \textit{io-bound}(\mathit{wload}(n)) \\ \mathit{false} & \textbf{if } \mathit{max}_m + \mathit{ram}(s) + \mathit{ram}(\mathit{wload}(n)) > \mathit{ram}(n) \\ \mathit{false} & \textbf{if } \mathit{mpl}(s) + \mathit{mpl}(\mathit{wload}(n)) > \mathit{cores}(n) \\ \mathit{true} & \textbf{otherwise} \end{cases}$$

In order to allow the scheduler to influence when and where which scripts are to be executed, it implements a windowing approach. When the scheduler is called to execute a script s, it will be added to the primary queue. All elements in this queue will be scheduled on a best-effort basis when the queue reaches a predefined maximum size (size_w, e.g., 10), or at least at a predefined interval following the last processing of the queue (time_w, e.g., 10 seconds). The scheduler processes the queue of scripts $\{s_i \mid 0 \leq i \leq k\}$ according to the order induced by $\mathit{priority}(s_i)$. A script s will be dispatched to a node n, which is currently able to process it without exhausting its resources, i.e., $\mathit{available}(n, s)$. When multiple such nodes are available, the scheduler dispatches the scripts according to the order induced over the nodes by $\mathit{priority}(n, s)$. The automatically generated scripts which export RDF data from primary sources, are an exception to this rule. They can not be scheduled to another node, because the primary source will in general only be reachable from a single node. When a script can not be dispatched to any node due to a lack of resources, it is removed from the primary queue and added to a secondary queue. The secondary queue is processed whenever resources become available, i.e., when the execution of a script has terminated. It is processed in the same way as the primary queue, which is shown in Algorithm 6.

6.2.4 Evaluation

For the evaluation of our scheduler we have implemented a complex scenario which closely resembles a real-world use case. An overview is shown in Figure 6.8. As can be seen, the system integrates four primary sources, including a Biobanking application, a Pathology Information System, a study extracted from a Clinical Data Management System (all based on relational databases) and ADT data from an HL7 message stream.

These original data sources are depicted by a blue color in the figure, their RDF representations are depicted by orange. Additionally, the system maintains metadata for the global dataspace (a global schema-level (TBox) and instance-level (ABox) ontology) as well as user-defined annotations (red) in native RDF triple stores. The primary and additional data sources are distributed over three sites, i.e., computing nodes. Nine data transformation scripts are utilized to process the data sources and to derive new knowledge or alternative representations (green). This includes the creation of a global Master Patient Index (MPI), which maps the local patient identifiers

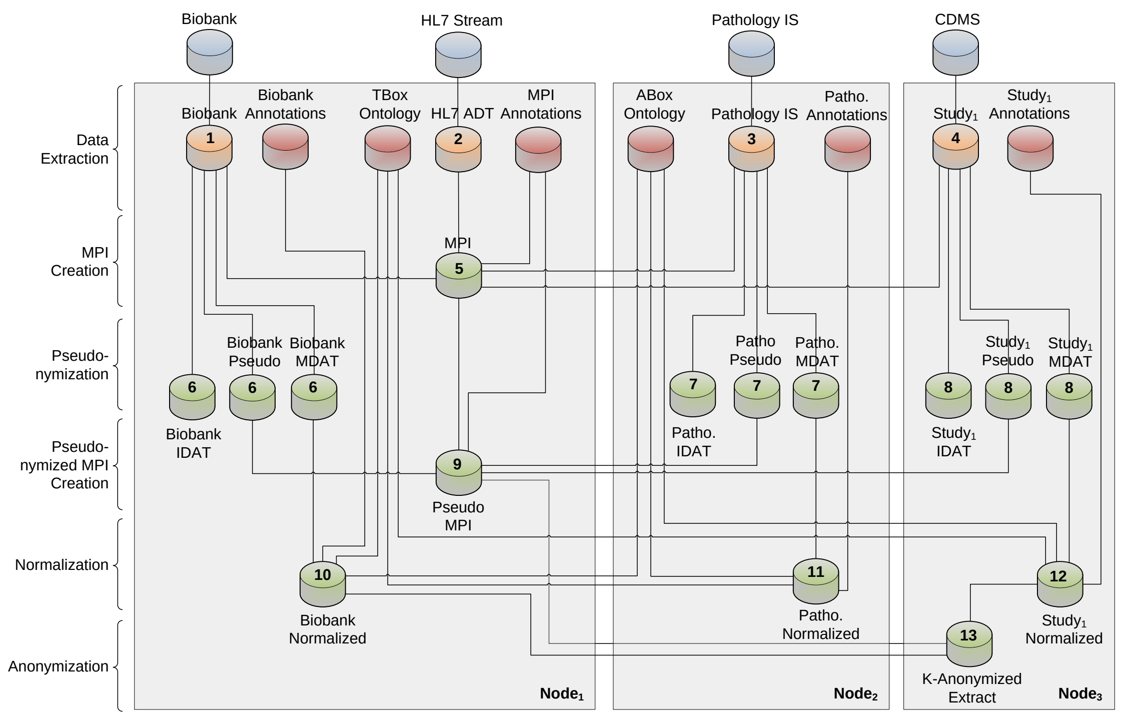


Figure 6.8: Example scenario

and takes user-defined annotations into account. Three pseudonymization steps are utilized to split the primary data into identifying and medical data for fine-grained access control. Semantic reasoning is utilized to derive normalized representations of the medical data from each primary source by taking user-defined annotations as well as global ontologies into account. Finally, a k-anonymized extract of an integrated view over the biobanking data and the study data extracted from the CDMS is created for external access. In our experiments, these operations are simulated by dedicated processes that exhibit realistic load profiles.

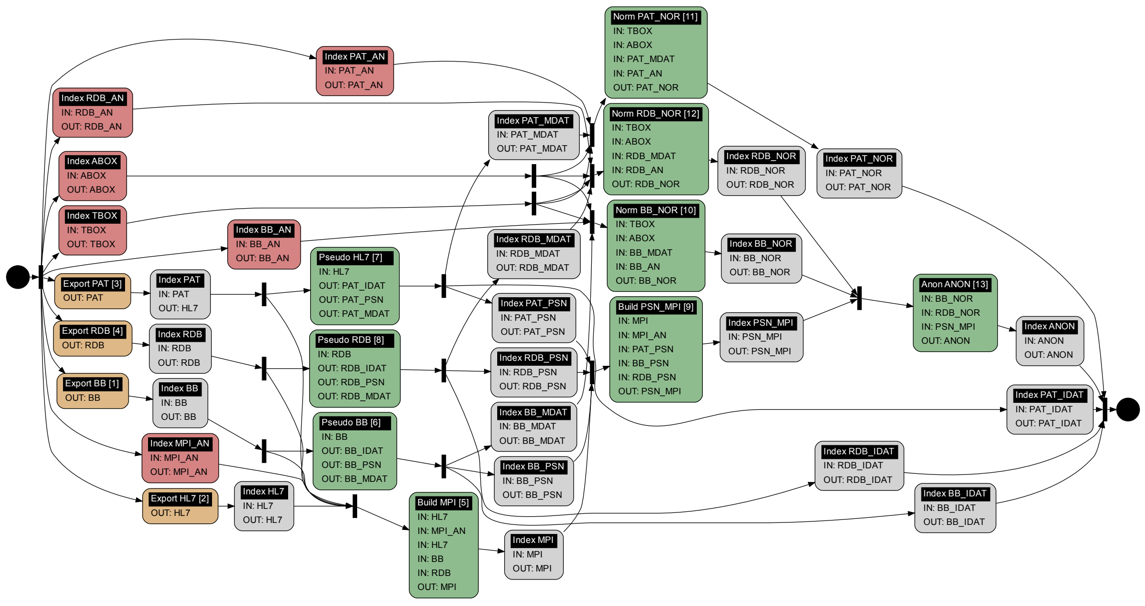


Figure 6.9: Example execution plan

Summarized, the scenario consists of four extracts from primary data sources, six native RDF repositories and 15 derived RDF datasets. In the resulting execution plan, which is shown in Figure 6.9, the system automatically inserted four scripts for exporting the data from the primary sources (orange) and 25 scripts which index all resulting RDF datasets (gray). In total, the execution plan therefore consists of 38 individual operations, which are to be executed in a distributed system with three nodes. The colors of the steps in Figure 6.9 resemble the colors from Figure 6.8.

The testbed equals the one utilized in Section 5.5 for benchmarking our distributed query processor. It consisted of three desktop machines, each of which was equipped with a 4-core 3.1 GHz Intel Core i5 CPU with 6 MB cache and 8 GB main memory, running a 64-bit Linux kernel in version 2.6.35. All workflows were executed on an Oracle JVM in version 1.7.0 with default settings. The systems were able to perform sequential reads on their local hard disks with about 100 MB/s and were connected via Fast Ethernet. The length of the window, $size_w$, was set to 10, and the timing interval, $time_w$, was set to 5 seconds. The maximum amount of utilized memory, max_m, was set to 6 GB, which is 75% of the local main memory. We show the feasibility of our scheduling approach by incrementally enabling the different heuristics.

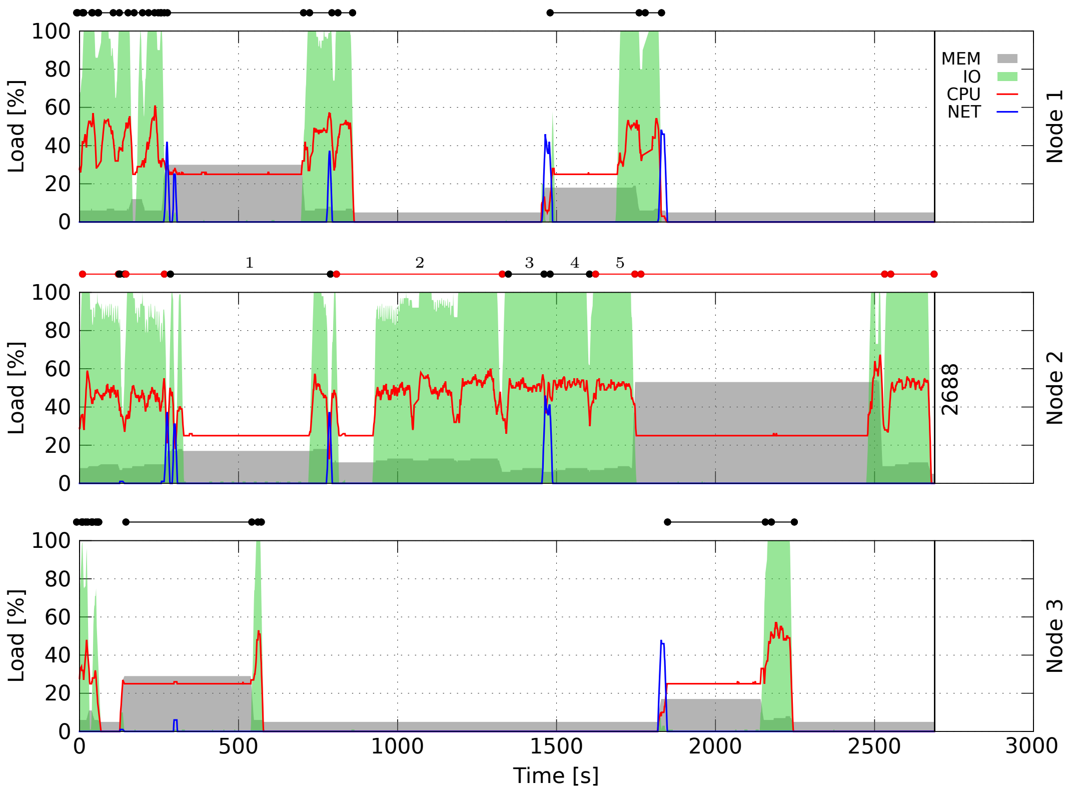


Figure 6.10: FIFO scheduling without local parallelism and local execution

Figure 6.10 shows the development of the different load parameters for all three nodes while executing the workflow with a basic version of our scheduler. Firstly, it processes the incoming execution requests in a FIFO manner, i.e., it omits sorting the scripts according to their priority (line 2 in Algorithm 6) and sets the maximum window size to $size_w = 1$. Secondly, it implements locally sequential execution, meaning that the function $available(n, s)$ will always return *false*, as soon as the node n is executing any script (line 4 in Algorithm 6). Finally, it allows only local execution, i.e., a script can only be dispatched to a node which hosts at least one of the accessed data sources. This is implemented by only considering nodes with $priority(n, s) \neq 0$ (line 5 in Algorithm 6). Each plot shows the local load for one node in terms of memory consumption, disk-IO load, CPU load and network traffic. Disk-IO and network-IO load are measured in terms of the utilized bandwidth. The spikes in the network load show, how the nodes exchange data during executing the workload. Above the plots, the lines indicate when a script is executed on a node. It can be seen that, in this configuration, the workflow engine needed 2683 seconds to execute the complete workload.

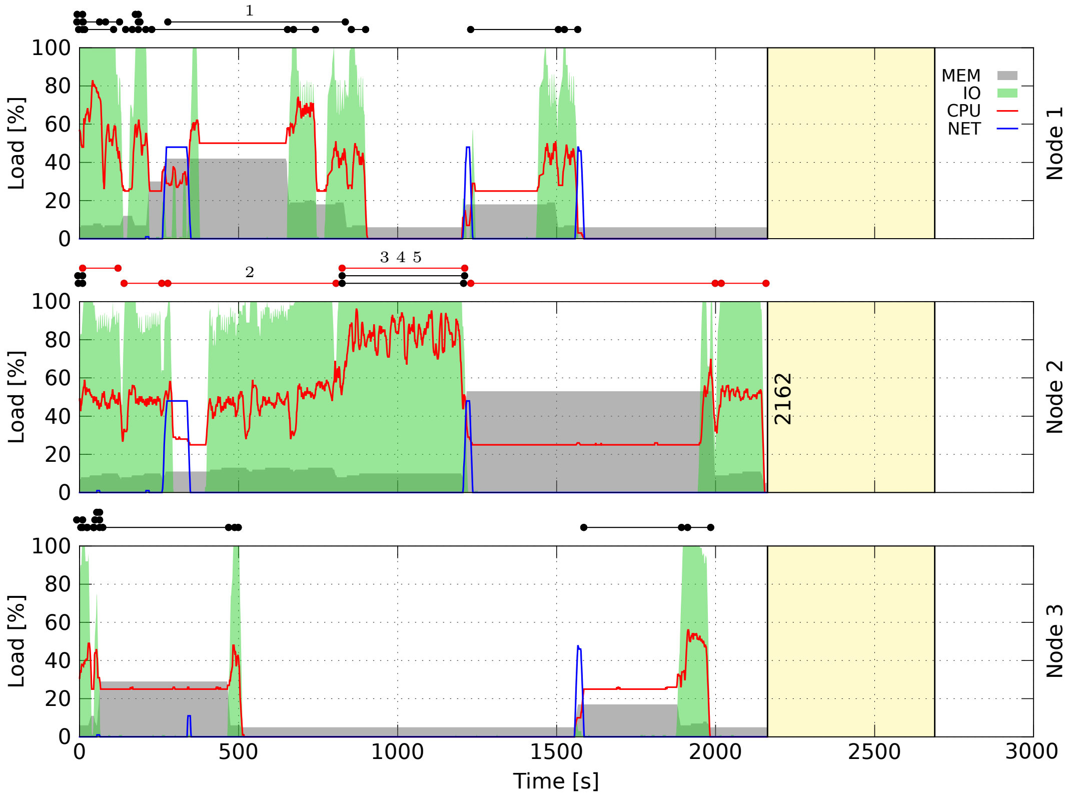


Figure 6.11: FIFO scheduling with local parallelism and local execution

The red lines indicate those scripts, which are part of the critical path and define a theoretical optimum for executing the workload. This theoretical optimum is determined by the longest path from the start-node of the execution plan to its end-node. In this context, the longest path means that sum of the execution times of all scripts executed on the path is maximal. The optimum is theoretical, because it is defined by a sequential execution of the workload, which assumes that there are no side-effects resulting from a parallel execution of the other (non-critical) paths. It can be seen that, despite some very short interruptions at the beginning, the path was mainly interrupted by the execution of the scripts annotated as 1, 3 and 4. It is important to note, that the execution times of this configuration are non-deterministic, because the scheduler processes the scripts on a first-come first-served basis and is called by several threads in parallel. The optimum for the complete workload is about 1882 seconds.

Figure 6.11 shows a configuration of the scheduler with enabled local parallelism. This means that each node is allowed to execute up to four scripts in parallel. As the scheduler is not able to prioritize scripts due to its FIFO strategy, we allowed for executing several disk-IO-bound processes in parallel in this scenario. It can be seen that the workload did strongly benefit from local parallelism, as the overall execution time dropped to 2162 seconds. Due to the parallelism, the critical path was not interrupted anymore, but the parallel execution of the scripts 3, 4 and 5 did not further decrease the execution times, as these three scripts had to share the local disk-bandwidth. Due to the parallelism, the script annotated as 1 was executed on node 1, which had a higher priority as it hosted more of its accessed datasets. In this configuration, more scripts were executed on a node with higher prioritity, which lead to a reduction in network communication.

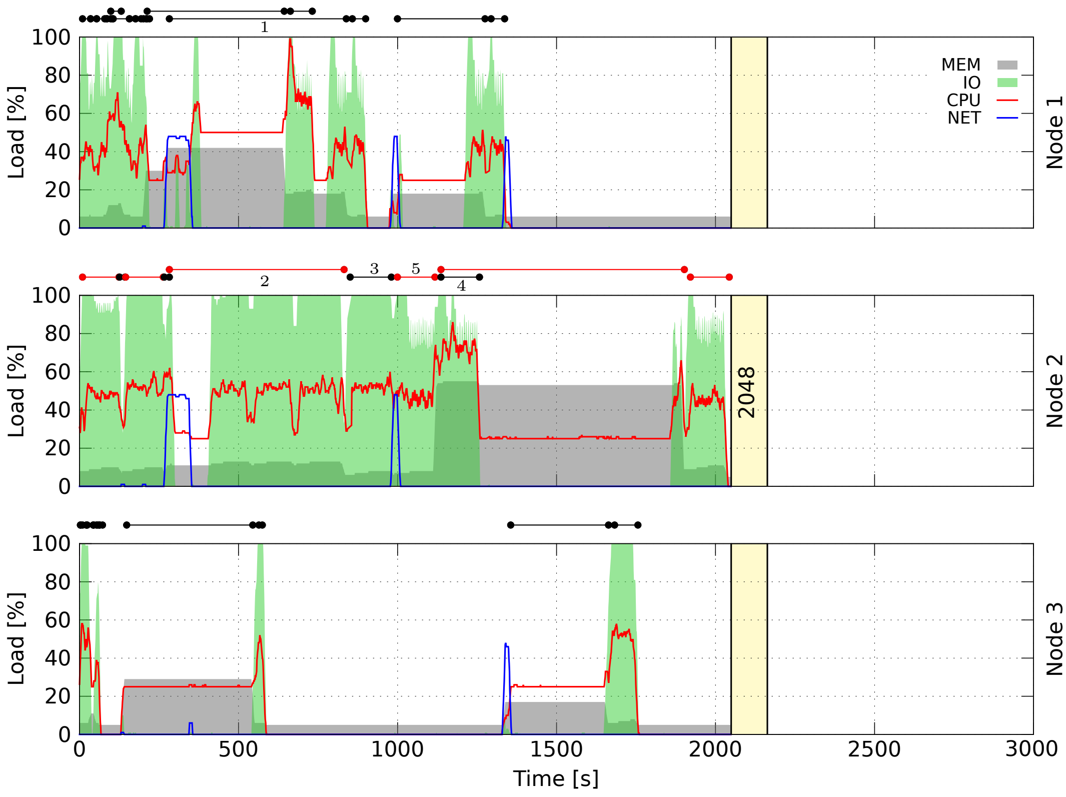


Figure 6.12: Windowed scheduling with local parallelism and local execution

Figure 6.12 shows a configuration of the scheduler which implements the windowing approach. This means, that it set $size_w = 10$ and prevented the concurrent execution of multiple disk-IO-bound processes on a local node, in order to allow for prioritization. As can be seen, the overall local parallelism decreased due to this restriction. Although this caused the interruption of the critical path with two IO-bound processes which could not be scheduled to another node any more, the overall execution time further decreased to 2055 seconds. The reason is, that the IO-bound script 5 could be prioritized over the IO-bound script 3, while script 4 was executed in parallel to a CPU-bound process.

Execution Mode	Overhead
FIFO, without local parallelism, without remote execution	42.8%
FIFO, with local parallelism, without remote execution	14.9%
Windowed, with local parallelism, without remote execution	8.8%
Windowed, with local parallelism, with remote execution	2.6%

Table 6.4: Comparison to the theoretical optimum (1882 s)

Finally, Figure 6.13 shows a configuration of the scheduler which implements all of the presented heuristics. In addition to the previous configuration, this scenario allowed the scheduler to dispatch scripts for execution on a remote node, i.e., a node which did not host any of the required data sources. It can easily be seen that this lead to an increase in network communication. As the time required for transferring relevant data between the nodes, is not dominant in the load profiles of standard processes, this still resulted in a decreased overall execution time of 1956 seconds. The critical step 5 was now dispatched to a remote node and that the critical path was not interrupted any more. There was, of course, an additional overhead introduced by transferring the required data to a remote node while executing the script.

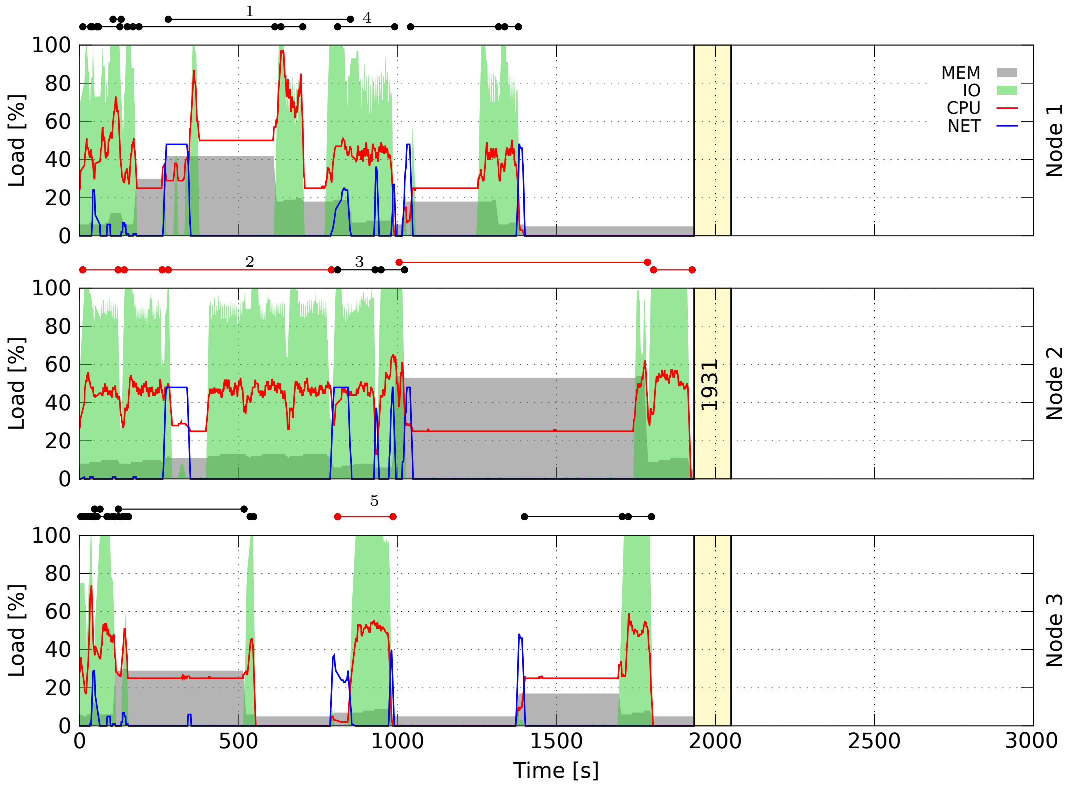


Figure 6.13: Windowed scheduling with local parallelism and global execution

Table 6.4 shows a comparison of the theoretical optimum with the execution times of all four configurations. It can be seen, that the overhead was reduced incrementally from 43.5% for the FIFO scheduler with locally sequential execution, to 4.6% for the windowed scheduler with parallel global execution. These numbers show, that the presented heuristics provide means to dynamically schedule scripts for data transformation and semantic integration in a way that achieves an efficient execution which leverages the available resources. The approach requires only very little knowledge about the characteristics of the individual steps and is therefore able to dynamically adjust to evolving runtime characteristics caused by changes in primary data.

6.3 Semantic Reasoning[4]

The remainder of this chapter presents different operators, which are provided for use in data transformation scripts. The manual annotation of data with semantic relationships can be utilized to automatically materialize implicit knowledge contained in the dataspace. Semantic reasoning techniques are highly valuable in our application domain, due to the inherent heterogeneity of the data involved.

This section first presents an overview over major vocabularies for formulating semantic relationships between type- and instance level data elements. It then presents

[4]Parts of the work presented in this section are based on the student project [vW12]

```
READS = { biobank, biobank-annotations }
WRITES = ∅
CREATES = { biobank-materialized }

\* Create a temporary database *\
Database temp = IO.createDatabase("temp", PERSISTENT);

\* Add data to the temporary database *\
IO.add("biobank-annotations", "temp");
IO.add("biobank", "temp");

\* Create the output database *\
Database DB_C = IO.createDatabase("biobank-materialized", "biobank", PERSISTENT);

\* Materialize implicit knowledge and add it to the dataset *\
TF.materialize("temp", "biobank-materialized");
```

Figure 6.14: Transformation script including OWL reasoning

and evaluates several semantic reasoners which can be utilized in data transformation scripts to perform semantic integration or normalization of heterogeneous data. For example, it can be stated that two resources with distinct identifiers describe the same entity, as in the following example:

```
(Patient1, diagnosis, Bronchial-Carcinoma)
(Patient2, diagnosis, Lung-Cancer)
```

The equivalence of both diagnoses can be made explicit by adding a triple which states that both of these resources describe the same entity:

```
(Bronchial-Carcinoma, same-as, Lung-Cancer)
```

Due to this additional information, it is also implicitly known that both patients have the same diagnosis. When this implicit knowledge is made explicit, i.e., materialized, the dataset would contain the following additional triples:

```
(Patient1, diagnosis, Lung-Cancer)
(Patient2, diagnosis, Bronchial-Carcinoma)
```

This section presents an overview over two major vocabularies, which build upon RDF and allow to express such semantic relationships in RDF data. The process of materializing the implied knowledge, which is called semantic reasoning, is supported by a transformation operator which relies on external software components. We evaluate different semantic reasoners for their suitability in this context. An according operator can be included in any of the transformation scripts described in the previous section. An example, which utilizes this operator is shown in Figure 6.14.

6.3.1 Vocabularies

The RDF/S Vocabulary

The RDF Schema (RDF/S) vocabulary allows to define a set-oriented class model, in which classes and properties are modelled separately [RDF]. It can be utilized to describe simple ontologies which support multiple inheritance for entities and properties and define a formal vocabulary for a certain domain or dataset. RDF/S separates all resources into groups, which are called classes. Each member of such a group is called an instance. Classes can be organized in a taxonomy, which is also possible for properties. Properties can be related to classes. The vocabulary also provides means to describe containers, collections and contains a reification vocabulary which allows to formulate statements about statements. In the following paragraphs we include a short overview over the expressiveness of this RDF vocabulary.

Important classes:

- `<rdfs:Resource>`: Defines a class for all RDF resources.
- `<rdfs:Class>`: Defines an abstract class for all classes.
- `<rdfs:Literal>`: Defines a class for all RDF literals.
- `<rdfs:Datatype>`: Defines a class for all typed literals.
- `<rdf:Property>`: Defines a class for all properties, i.e., predicates.

Important properties:

- `<rdfs:subClassOf>`: Allows to define a class hierarchy. All instances of the subclass are also instances of the superclass.
- `<rdfs:subPropertyOf>`: Allows to define a hierarchy of properties. All RDF nodes which are related via the subproperty are also related via the superproperty.
- `<rdfs:range>`: Defines the class of allowed objects of a property.
- `<rdfs:domain>`: Defines the class of allowed subjects of a property.
- `<rdf:type>`: States that the subject is an instance of the class object.
- `<rdfs:label>`: Defines a human-readable name of a resource.
- `<rdfs:comment>`: Defines a human-readable description of a resource.

As can be seen, the RDF/S vocabulary only offers limited ways to fully conceptualize an application domain but is more oriented towards a definition of the schema of a dataset in terms of instances, classes and their relationships. The most important application for semantic reasoners in this context is the materialization of the transitive closure of a dataset's class and property hierarchies.

The OWL Vocabulary

In contrast to RDF/S, the Web Ontology Language (OWL) is a much more expressive RDF vocabulary for the specification of ontologies [OWLa]. Semantic reasoners for OWL provide different functionalities, such as consistency checking, deciding concept membership, answering conjunctive queries, classification or rule support. It builds upon RDF/S and provides additional means for formulating statements similar to first-order logic. OWL 2 is a backwards-compatible successor of OWL 1 and specifies three different tractable fragments, called OWL 2 Profiles [OWLb]. OWL 2 EL enables PTime algorithms for all common reasoning tasks, OWL 2 QL enables conjunctive queries to be answered in LogSpace and OWL 2 RL allows to implement all common reasoning taks in PTime with rule engines. In turn, OWL 1 consists of three layered subsets:

OWL Lite contains a limited subset of OWL. It allows to formulate concept-hierarchies as well as simple boundary conditions. This includes, e.g., cardinality restrictions, which are limited to cardinalities of zero or one. On this layer, OWL reasoning is much less complex than on the other layers.

OWL DL allows for a maximum of expressiveness, but still guarantees computability and decidability. It contains all OWL constructs, but restricts their use in some contexts. E.g., a class can not be an instance of another class if it is a subclass of more than one classes. The term "DL" stands for *description logics* and OWL DL thus implements a decidable subset of first-order logic.

OWL Full provides maximum expressiveness and syntactic freedom, but does not guarantee computability. It, e.g., allows an ontology to extend the predefined vocabulary and is the only OWL version that contains the complete RDF/S vocabulary. Due to its limitations it is only rarely used in practice.

The following paragraphs give a quick overview over the parts of the OWL vocabulary, which are most important in our context. For a deeper insight into other aspects, e.g., the definition of cardinality restrictions for consistency checking, the interested reader is referred to [OWLa].

Important class definitions:

- `<owl:Class>`: A class defines a group of individuals which share common properties.
- `<owl:Individual>`: Individuals are instances of classes. Properties are utilized to relate individuals.
- `<owl:intersectionOf>`: Allows to define classes with instances from an intersection of sets of individuals.
- `<owl:unionOf>`: Allows to define classes with instances from a union of sets of individuals.

Properties for expressing equivalences:

- `<owl:equivalentClass>`: Equivalent classes share the same instances. Can be utilized to model synonyms.
- `<owl:equivalentProperty>`: Can be utilized to model synonyms.
- `<owl:sameAs>`: Expresses the equivalence of two individuals.
- `<owl:differentFrom>`: States that two individuals are not the same.

Properties and classes for characterizing predicates:

- `<owl:ObjectProperty>`: A subclass of `<rdf:Property>` which defines a relationship between individuals.
- `<owl:DatatypeProperty>`: A class for properties which define a relationship between an individual and a literal.
- `<owl:TransitiveProperty>`: A class for transitive properties.
- `<owl:SymmetricProperty>`: A class for symmetric properties.
- `<owl:FunctionalProperty>`: A class for properties that have only one (unique) value for each individual.
- `<owl:InverseFunctionalProperty>`: A class for properties whose value uniquely determines the subject.
- `<owl:inverseOf>`: States that a property is the inverse of another property.

6.3.2 Evaluation

Semantic reasoning is one of the core functionalities of an ontology-based integration system. While reasoners normally have to deal with typical TBox (e.g, satisfiability, subsumption checks) or ABox (e.g., consistency checks, conjunctive querying) reasoning tasks, our system requires a different setup. In order to include implicit knowledge into the dataspace in a sound way, a reasoner is required to materialize all inferred statements as specified by an input dataset. As a result, the enriched dataset can be indexed and queried analogously to all primary data. This functionality is not extensively covered by previous work which aimed at evaluating different OWL and RDF/S reasoners, such as [GPH04, GPH05, GTH06, BHJV08, DCtTdK11, LYH12] and [KLK12]. This is because this task is normally considered impractical, as full reasoning over complex ontologies can lead to excessive resource consumption. As large parts of the inferred knowledge will never be required, e.g., for answering conjunctive queries, most reasoners will perform semantic reasoning only on demand. Nevertheless, the full-materialization approach is implemented by some reasoners, e.g., [OWLc] and partially [HER].

In the context of the integration system presented in this thesis, this approach is practical, though. The idea of implementing semantic integration and data transformation in a punctual and incremental manner for these parts of the dataspace for which it is required is inherent to the underlying concept. To this end, the presented scripting environment and querying engine provide means to extract exactly those parts of the data for which semantic reasoning is to be performed. This pay-as-you-go approach heavily reduces the resource consumption, e.g., compared to materializing all implicit knowledge in the dataspace. Moreover, the application of separate, encapsulated semantic reasoning tasks to different parts of the data, allow to paralellize their execution to a large extent. Finally, the system filters the resulting implicit knowledge for redundant basic axioms, which are of little use to the users but might consume a lot of resources (e.g., every resource is distinct from nothing). The same approach is, e.g., implemented by [GMF+03]. There are some OWL constructs, though, which result in an exponential growth of the implicit knowledge in a dataset and should be avoided [WLL+07].

As our requirements are not extensively covered by previous work, we have evaluated several reasoners in order to decide which reasoner is suited best for including it into our semantic integration component. Because OWL is much more expressive than RDF/S, we focus on OWL reasoning. To cover different aspects of semantic integration tasks, we performed benchmarks oriented towards instance data and benchmarks focussed on schema-level descriptions.

Semantic Reasoners and Setup: We evaluated OWL reasoners that come with an open source or dual license and with which we were able to implement the full-materialization approach. This includes the Pellet reasoner [PEL] which implements OWL DL reasoning (with some minor limitations) in Java and utilizes the well-known *Tableau* algorithm. The HermiT reasoner is a Java-based open source OWL DL reasoner which implements an algorithm called *Hypertableau* [HER]. Fact++ is an open source C++ implementation of the Fact OWL DL reasoner which also implements the Tableau algorithm. Other reasoners, such as OWLIM [OWLc] and the Apache Jena project, implement a rule-based approach. Although providing OWL reasoning via a rules engine is rather simple to implement, the systems are only able to process less expressive subsets of OWL than the other reasoners and suffer from some performance limitations. The Jena framework therefore includes different rule-based reasoners which are able to handle different subset of OWL Full. Because these different levels of expressiveness offer different levels of performance (*full*,*mini* and *micro*) we have chosen this reasoner for our benchmarks. The rules included in Jena Full are incrementally reduced in order to reduce computational complexity. For example, support for the <*owl:sameAs*> predicate is only included in Jena Full and Jena Mini but not in Jena Micro. There are other well known reasoners which are not included in our evaluation, such as the KAON2 reasoner [MS05] which implements a novel strategy by efficiently reducing reasoning processes to recursive datalog programs. We were not able to implement the full-materialization approach with the API provided by KAON2.

Ontology	TBox	Class	SubC	Property	SubP	Vocabulary	ABox	Scale
LUBM	293	43	36	31	5	OWL Lite	8.519	1
							15.143	2
							21.415	3
							27.794	4
							34.550	5
Vicodi	118.419	167	167	8	5	RDFS(DL)	54.081	1
							107.734	2
							161.387	3
							215.040	4
							268.693	5
SWRC	1.817	114	245	127	6	OWL DLP	5.379	1
							12.382	2
							16.710	3
							20.465	4
							25.781	5
Wine	1.936	142	126	13	5	OWL DL	3.265	1
							6.005	2
							8.977	3
							12.181	4
							15.617	5

Table 6.5: Datasets utilized for ABox reasoning

We distinguished between two different scenarios, ABox and TBox reasoning. The ABox reasoning benchmarks model scenarios in which a generic OWL DL reasoner materializes implicit knowledge about instance data (ABox) of increasing size for a fixed schema-level description (TBox). This scenario is the most important for semantic integration purposes, as this mostly requires the materialization of relationships between instance data. In some contexts, the integration of complex schema-level descriptions might also be relevant. This is modelled by our TBox reasoning benchmarks. Here, the reasoners need to classify complex class hierarchies(i.e., materialize implicit schema-level descriptions such as inheritance). Some highly specialized solutions also exist for TBox reasoning [DCtTdK11], but we have chosen generic OWL DL reasoners as the reasoners will need to perform integrated ABox and TBox reasoning.

All experiments were performed on a desktop machine with a 4-core 3.1 GHz Intel Core i5 CPU with 6 MB cache running a 64-bit Linux kernel in version 2.6.35. All reasoners were executed on an Oracle JVM in version 1.7.0 with default settings and a heap size of 4 GB. For each reasoning task we set a timeout of 120 minutes. The reported execution times include importing, reasoning and exporting the datasets, as some reasoners perform excessive pre-processing and on-demand reasoning. We compared Jena 2.6.4, Pellet 2.3.0, HermiT 1.3.6 and Fact++ 1.6.0.

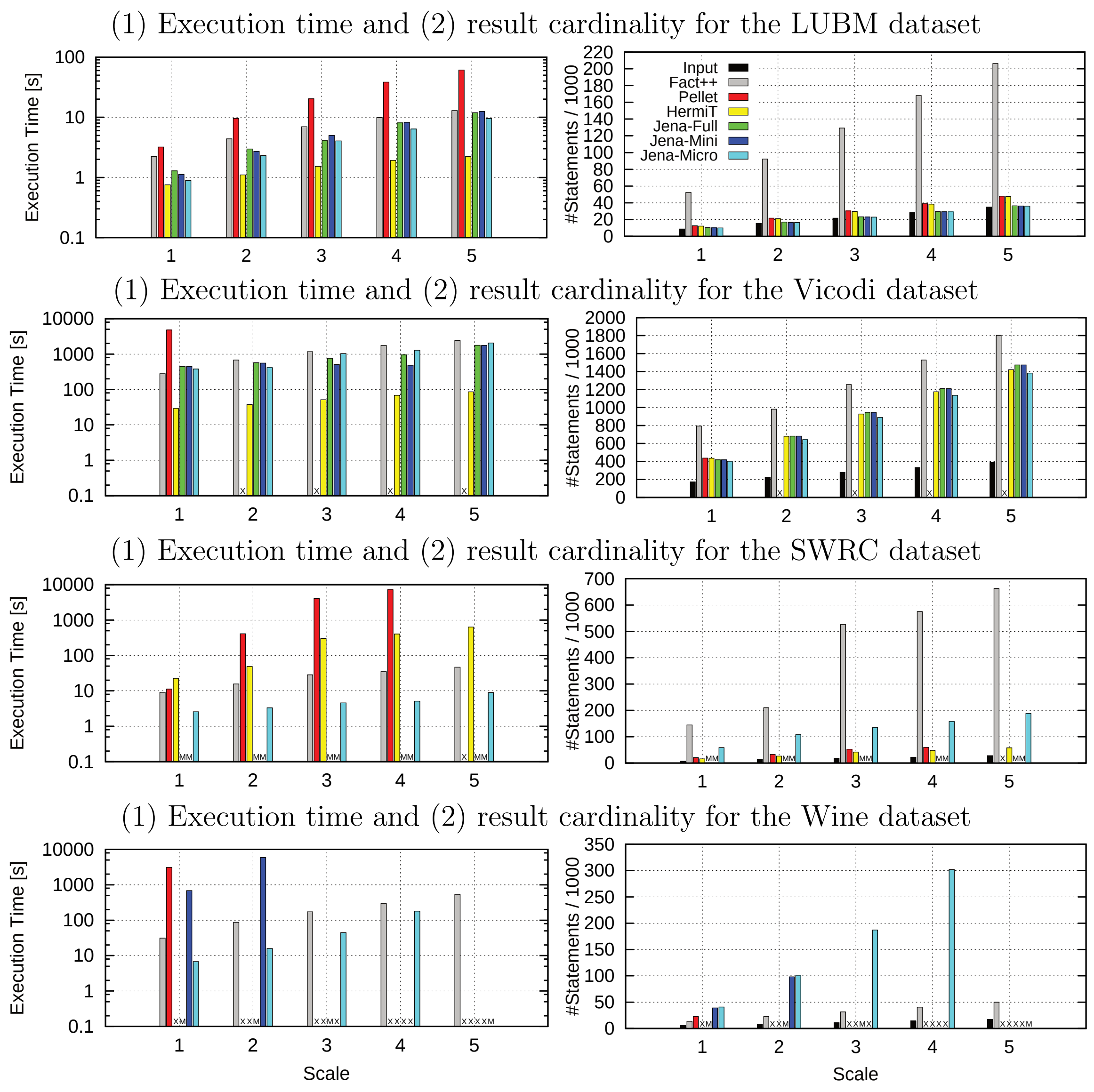


Figure 6.15: Materializing implicit knowledge for an ABox of increasing size

ABox Reasoning: Here, we have chosen four different datasets which have also been utilized for benchmarking OWL reasoners in [GPH05, BHJV08] and [LYH12]. Each of the datasets consisted of a TBox of constant size and an ABox with linearly increasing size. Additionally, the datasets utilize different fragments of OWL that allow for different levels of expressiveness and thus different levels of computational complexity. An overview over the basic properties of the datasets is shown in Table 6.5. To give an idea of the complexity of the ontologies, we include the overall size of the TBox, the number of contained classes, properties, `sub-class-of` and `sub-property-of` relationships as well as the size of the ABox for every scale. The Vicodi dataset utilizes the least expressive vocabulary, RDFS(DL). It allows to express schema-level knowledge similar to RDF/S but utilizes a subset of the OWL DL vocabulary. On the other hand, Vicodi is by far the largest dataset, comprising a TBox of over 100.000 statements and ABoxes of up to more than 250.000 statements. The Wine dataset comprises the ABox with the fewest statements (up to around 15.000 statements) but utilizes the most expressive TBox, with around 2.000 statements that fall into OWL DL. Both LUBM and SWRC utilize vocabularies whose expressiveness lies in between RDFS(DL) and OWL

DL and ABoxes of medium size. The TBox of the SWRC dataset is more than six times bigger than the TBox of the LUBM dataset. Despite LUBM which has a rather small TBox, the datasets' class hierarchies are of comparable size.

The results of our evaluation in terms of execution times and the number of generated statements are shown in Figure 6.5. We indicate a timeout by *"X"* and a termination because of the main memory limit by *"M"*. It can be seen that all reasoners perform well on the LUBM dataset and the complexity steadily increases when processing the Vicodi, SWRC and Wine datasets. The number of timeouts increases from SWRC, Vicodi to Wine. This complexity does not correlate with the size of the datasets (ABox or TBox) or the complexity of the class hierarchy, but with the expressiveness of the utilized OWL vocabulary. Although the Vicodi dataset utilizes the least expressive OWL subset, its results stand out, because its ABox is an order of magnitude larger than the other datasets. The fastest reasoner was able to process the datasets in scale 1 in about 1s for LUBM, 29s for Vicodi, 9s for SWRC and 31s for Wine. It can be seen that Fact++ is by far the most efficient OWL DL reasoner in our tests, followed by HermiT. Vicodi is the only dataset in which HermiT outperforms Fact++. Pellet seems to offer the worst overall performance, but its results are not always comparable to the Jena reasoners. In contrast to the other reasoners, the Jena reasoners only process subsets of statements from OWL DL. In some cases they are therefore able to process a dataset before reaching a timeout, while other reasoners (e.g., Pellet) reach the timeout limit due to the more complex reasoning task. We have included the Jena reasoners into our evaluation, because they might be suitable for some reasoning tasks and their performance is thus relevant.

Ontology	TBox	Class	SubC	EquiC	Property	SubP	Overlap	Scale
Gene Ontology	15.140	1.518	1.581	19	3	11	–	1
	30.148	2.921	3.425	34	4	11	12%	2
	45.366	4.318	5.390	79	4	11	15%	3
	60.694	5.761	7.487	188	6	13	16%	4
	76.047	7.095	9.534	239	7	14	22%	5
GALEN Ontology	15.056	2.458	1.655	430	306	312	–	1
	30.056	5.361	4.344	896	382	388	0%	2
	45.072	8.734	6.572	1.480	470	476	11%	3
	60.075	11.971	8.539	2.072	510	518	13%	4
	75.076	15.347	10.589	2.745	542	546	15%	5
NCI Thesaurus	50.011	2.913	4.989	0	32	0	–	1
	100.039	5.960	9.891	0	36	0	18%	2
	150.048	8.856	15.055	0	42	0	21%	3
	200.067	11.949	19.954	0	44	0	23%	4
	250.067	14.966	25.161	0	48	0	25%	5

Table 6.6: Datasets utilized for TBox reasoning

Although all reasoners despite Jena implement the most important parts of the OWL DL vocabulary, they support slightly different subsets. This is also reflected by the differing numbers of statements created when processing the benchmark datasets as can be seen in Figure 6.15. The number of statements materialized by the Jena reasoner decreases with the capabilities of the specific implementation (full, mini or micro). Despite that, the differences in the resulting cardinalities mainly show the different basic axioms which are generated by the reasoners (e.g., each class is a subclass of itself). As these statements are presumably not relevant to our use cases, they can and should be removed from the resulting datasets. After this clean-up process, all reasoners which implement OWL DL, produce roughly the same results. There are some slight differences, though, e.g., if a reasoner generates OWL 2 output.

TBox Reasoning: For our evaluation of the systems' TBox reasoning capabilities we chose three biomedical ontologies, which lie in the OWL 2 EL fragment. OWL EL is an important subset of OWL 2, which is sufficiently expressive to cover many biomedical ontologies and allows the implementation of polynomial algorithms for all standard reasoning tasks [OWLb, DCtTdK11]. Important ontologies in this fragment include SNOMED CT, the Gene Ontology, the NCI Thesaurus and the GALEN Ontology, from which we have chosen the latter three for our benchmarks.

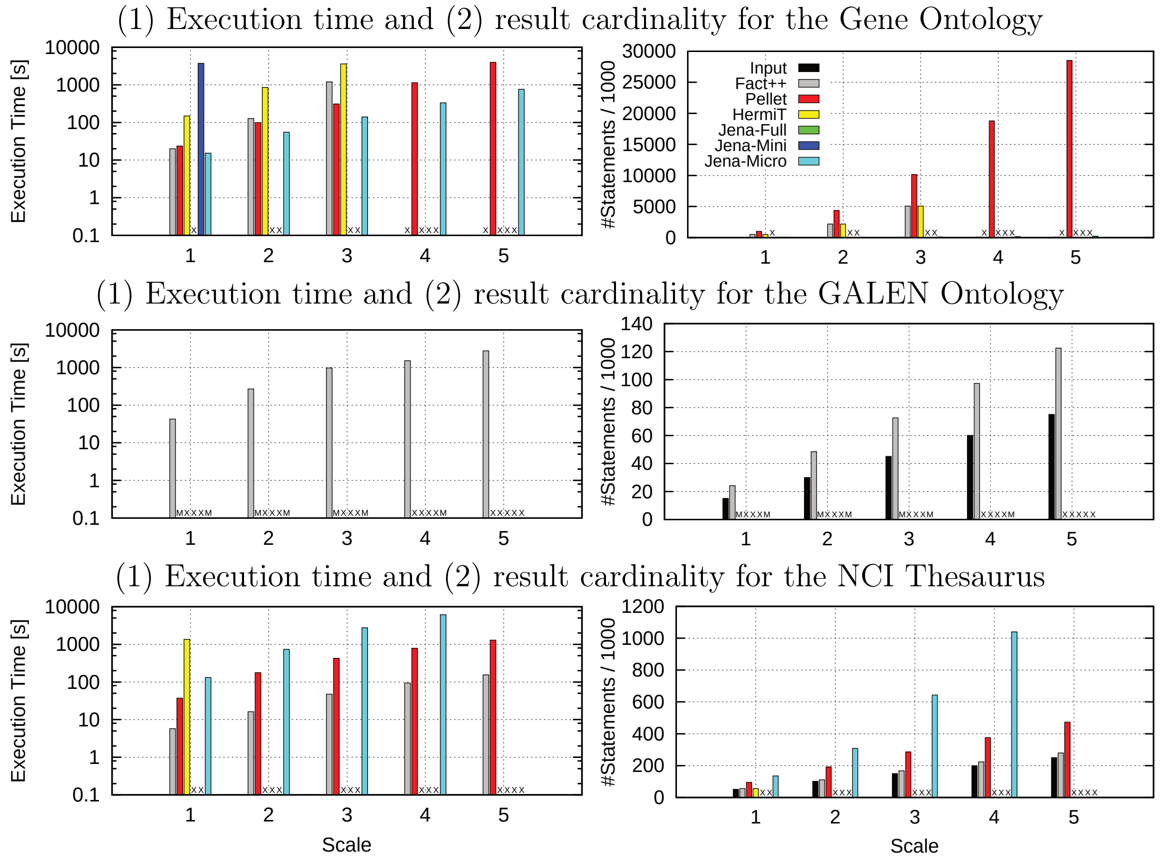


Figure 6.16: Materializing implicit knowledge for a TBox of increasing size

An overview over the basic properties of these datasets is shown in Table 6.6. In addition to the properties presented for the ABox evaluation datasets, we provide the number of `equivalent-class` relationships. On average, the datasets are larger and model much more complex class definitions as the datasets utilized in the ABox benchmarks. In order to derive datasets of different scales, TBoxes can not be simply split into disjunct subsets, as these might be inconsistent. The process of splitting ontologies into smaller subsets is called *Segmentation* or *Modularization*. Dedicated algorithms normally start at a specific concept and extract all knowledge which is relevant in this context. Previous work has proposed different types of algorithms which interpret the data to derive complete results, e.g., [SR06] and [GHKS08]. As we wanted to extract lightweight subsets, which do not contain any knowledge that was not already contained in the original dataset, we implemented a different approach. Starting from each concept in the ontology, we derived a subset by including all resources and statements which could be reached by traversing the underlying directed RDF graph. In this way, the resulting subsets are consistent but might lack some additional information, e.g., from `equivalent-class` definitions that were defined by an incoming edge to any of the contained concepts. The union of all subsets exactly matches the original dataset. We have merged the individual subsets to derive five different scales for each dataset. The complexity of the subsets does not only increase due to the increasing size, but also because the subsets of the class hierarchy share some concepts and the complexity of the resulting hierarchy thus increases with every additional subset. This is indicated by the *overlap* property in Table 6.6, which shows how much of the concepts in an individual scale were already contained in the previous scale. At scale five, the subsets' sizes compared to the original datasets is ca. 5% for the Gene Ontology, 30 % for the Galen Ontology and 50% for the NCI Thesaurus. The GALEN dataset is considerably more complex than the other datasets [GHKS08].

The results are shown in Figure 6.16. Similar to the other scenario, Fact++ offers the best performance and is the only reasoner which is able to handle at least some scales of all datasets. Pellet offers the second best overall performance and is the only OWL DL reasoner that can handle all scales of the Gene Ontology. The Hypertableau algorithm implemented by HermiT seems to be not very well suited for this scenario. Jena-Micro performs reasonably well, but interprets only some of the contained statements. In these benchmarks, the size and complexity of the tested ontologies clearly takes the reasoners to their limits. Although TBox reasoning is relevant, the classification of large and highly complex class hierarchies is unlikely to be a requirement for real-world integration scenarios, as these normally require the materialization of dependencies between individuals and classes from different datasets. This is more closely resembled by our ABox benchmarks.

The presented benchmarks are not intended to provide an exhaustive evaluation of semantic reasoners for OWL, but to give hints on which reasoners are suited best for fully materializing implicit knowledge for semantic integration purposes. In this context, all of the OWL DL reasoners contained in our comparison support the most important statements, i.e., defining class hierarchies with multiple inheritance for entities and properties and for formulating the relationships between schema-level and instance-level data elements. Many reasoners generate a large amount of redundant statements. The utilization of open source reasoners allows to tailor their implementation to the specific functionality required by the system (full materialization) and prevent redundant statements from being generated. Some reasoners also provide specific interfaces for this tasks, e.g., Pellet. Fact++ offers by far the best overall performance. Due to the differing capabilities we have included Fact++ and the Pellet reasoner into our framework. By default, semantic reasoning is performed by Fact++, whereas Pellet can be utilized in the rare cases where it is better suited for the requirements. In [DCtTdK11] a comprehensive overview over scalable TBox reasoners is presented, which can be utilized if large-scale TBox reasoning is required.

6.4 Data De-Identification

The de-identification of sensitive datasets is an important technique for protecting an individuals' privacy when integrating data [Gar01]. Different incidents (e.g., [NS08, ZB06, Swe01]) have shown that simply removing all directly identifying information (e.g., a persons name) is not sufficient to protect an individual's privacy. In addition, researchers need high-quality and fine-grained data collections. In this context, anonymization is an important building block for balancing the individuals' privacy and the researchers' needs. To this end, *k-anonymity* is a wide-spread technique. The basic idea is to protect a dataset against re-identification by generalizing and suppressing the *quasi identifiers*, which are the attributes that could be used in a linkage attack. This attack tries to link anonymized data to additional identified data, which can result in identity disclosure [Swe01]. A dataset is considered *k-anonymous* if each data item can not be distinguished from at least $k - 1$ other data items [SS98]. An example dataset with quasi identifiers age, gender and zipcode as well as a two-anonymous transformation are shown in Figure 6.7. Without the loss of generality we assume a tabular data structure. Although various other methods are currently under discussion (e.g., differential privacy [Dwo06]), k-anonymization is still considered the option of choice for in many domains, especially in the area of medicine [DE12].

Age	Gender	Zipcode
34	female	81667
45	female	81675
66	male	81925
70	male	81931
34	female	81931
70	male	81931
45	female	81931

Age	Gender	Zipcode
]25-50]	female	816⋆⋆
]25-50]	female	816⋆⋆
]50-75]	male	819⋆⋆
]50-75]	male	819⋆⋆
]25-50]	female	819⋆⋆
]50-75]	male	819⋆⋆
]25-50]	female	819⋆⋆

Table 6.7: Example dataset

Within our platform for incremental ontology-based integration, data de-identification is provided as an operator, which can be included into the transformation scripts presented in the previous section. An example is shown in Figure 6.17. The script implements a basic data pseudonymization and de-identification process. Before the script is executed, the database *biobank* contains identified data of individuals. For this example, we assume the presence of the attributes *gender*, *zipcode* and *date of birth (dob)* and the existence of a class *Patient*, which is instantiated by resources describing individuals. When executed, the script creates two more databases. The database *biobank-anonymized* contains an anonymized and pseudonymized version of the original dataset. Firstly, it is ensured that the data in *biobank-anonymized* is 5-anonymous over the contained attributes and that all URIs identifying patients are replaced with a pseudonym. Secondly, the database *biobank-pseudonymized* contains a mapping from the pseudonyms to the original identifiers.

6.4.1 Related Work

Generally, there are multiple ways to transform a dataset into a k-anonymous representation and different algorithms have been proposed. These can be classified according to different axes. Firstly, some algorithms implement global recoding (also called full-domain anonymization), whereas others implement local recoding. *Local recoding* means that, within a column, different generalization rules can be applied to equal values, whereas *global recoding* guarantees to apply the same rule. Local recoding is often used by clustering algorithms (e.g., [APF$^+$10]), whereas global recoding is mostly used by algorithms which utilize generalization hierarchies. Some algorithms do not find an *optimal solution*, whereas others do. Optimal is defined as the solution which results in minimal information loss according to a given metric. Solving this problem has been proven to be NP-hard [MW04]. Some algorithms (e.g., [AFK$^+$05]) approximate the problem and only find a solution that is within a guaranteed distance to the optimum. Various extensions to the k-anonymity concept exist. The most prominent examples are

```
READS = { biobank }
WRITES = ∅
CREATES = { biobank-anonymized, biobank-idat, biobank-pseudonyms, biobank-mdat}

\* Define pattern describing individuals *\
String PATTERN =  "?id <rdf:type> <biobank:Patient>.
                   ?id <biobank:patid> ?patid.
                   ?id <biobank:gender> ?gender.
                   ?id <biobank:dob> ?dob.
                   ?id <biobank:zipcode> ?zipcode.";

\* Create the output databases *\
Database anonymized = IO.createDatabase("biobank-anonymized", "biobank", PERSISTENT);
Database idat = IO.createDatabase("biobank-idat", "biobank", PERSISTENT);
Database pseudonyms = IO.createDatabase("biobank-pseudonyms", "biobank", PERSISTENT);
Database mdat = IO.createDatabase("biobank-mdat", "biobank", PERSISTENT);

\* Pseudonymize the data *\
TF.pseudonymize(  "biobank", "select * where {"+PATTERN+"}",
                  "biobank-idat", "biobank-pseudonyms", "biobank-mdat");

\* Execute pattern on the medical database *\
rs = IO.executeQuery(      "biobank-mdat",
                           "select * where {"+PATTERN+"}");

\* Anonymize the temporary data with DM* metric" *\
rs = TF.anonymize(rs, 5, 0.0d
                  TF.getHierarchy(TF.GENDER, "gender")),
                  TF.getHierarchy(TF.DOB, "dob")),
                  TF.getHierarchy(TF.ZIPCODE, "zipcode")));

\* Store the anonymized and pseudonymized data in "biobank-anonymized" *\
Database anon = IO.createDatabase("biobank-anonymized", "biobank", PERSISTENT);
IO.construct(anon, rs, PATTERN);
```

Figure 6.17: Transformation script including data de-identification

l-diversity [MKGV07], t-closeness [LLV07] and d-presence [N+07]. To derive appropriate parameters for the algorithms risk based anonymization [Ema10] can be used. An overview of other privacy models and algorithms can be found in [CdVFS08,FWCY10]. Without loss of generality we focus on the basic k-anonymity problem.

In [EDI+09] it has been shown that algorithms, which find an optimal solution by applying global recoding with user-defined generalization hierarchies, are to be preferred in the biomedical domain. The main reasons for this are that an optimal solution guarantees minimal information loss, which is important for the usefulness of the result. Furthermore, global recoding delivers the best result in terms of statistical properties. Finally, the utilization of user-defined generalization hierarchies allows to define different generalization strategies for different use-cases. [LQS11] even showed that algorithms based upon generalization hierarchies can, under certain circumstances, guarantee a better degree of privacy than the ones based on clustering and local recoding.

Generalization and Monotonicity

Generalization *hierarchies* define a way of iteratively generalizing the values of an attribute. Figure 6.18 shows generalization hierarchies for the quasi identifiers in the example dataset. In the remainder of this section we assume that the height of the hierarchy for the attribute *age* is 2. In the two-anonymous version from Figure 6.7 the age is transformed to intervals of length 50 (level 1), the attribute gender is suppressed (level 1) and the two least significant digits are dropped from the zipcode (level 2).

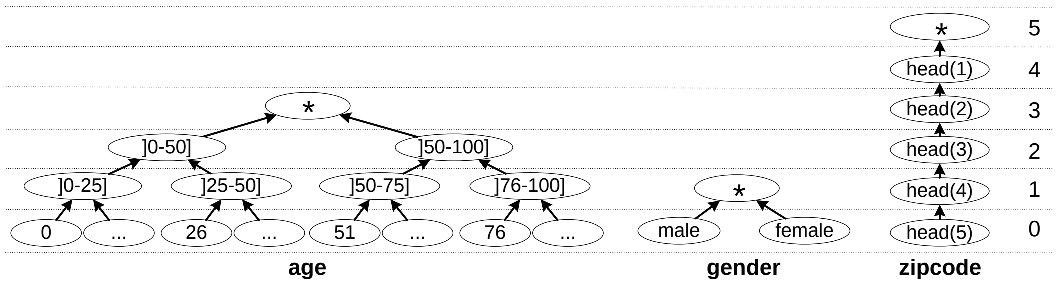


Figure 6.18: Generalization hierarchies

Most algorithms which utilize generalization hierarchies operate on a data structure called *Generalization Lattice*. It is shown in Figure 6.19 for our example. An arrow denotes that a state is a direct *generalization* of a more *specialized* state and can be created by incrementing the generalization level of one of its predecessors' quasi identifiers. The state with minimal generalization $(0, 0, 0)$ is at the bottom and represents the original dataset, whereas the state with maximal generalization $(2, 1, 5)$ is at the top. The state that has been applied to the dataset from Figure 6.7 is $(1, 1, 2)$.

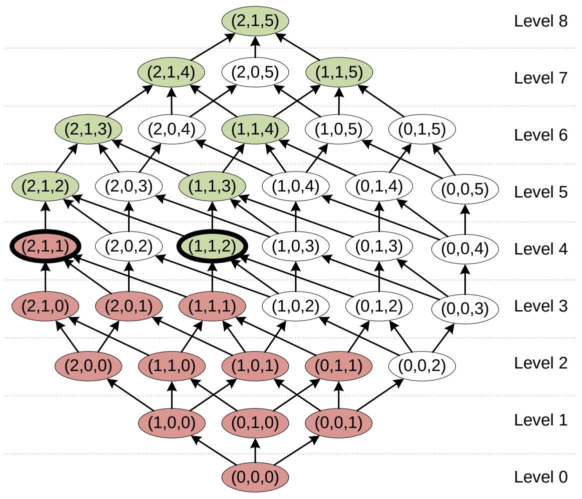


Figure 6.19: Lattice and predictive tagging

Monotonicity is a very important property, which enables several optimizations for globally optimal k-anonymity algorithms. Firstly, the authors of [LDR05] have introduced the notion of *monotonic generalization hierarchies*. In a monotonic generalization hierarchy the groups at level $l + 1$ are built by merging groups from level l. This allows prune large parts of the search space, because all states which are successors of an anonymous state are also anonymous. Moreover, all predecessors of a non-anonymous state are also non-anonymous. This is because generalization is monotonic for the complete dataset, if it is monotonic for each quasi identifier. An example is shown in Figure 6.19, where the fact that $(2, 1, 1)$ is non-anonymous implies that all of its predecessors are also non-anonymous (dark gray). In addition, all successors of the anonymous state $(1, 1, 2)$ are also anonymous (light gray). Secondly, the authors of [EDI$^+$09] proposed to utilize *monotonic metrics*. The monotonicity criterion for metrics requires that on each path from the bottom node to the top node of the generalization lattice the values for information loss increase monotonically. This implies that if a generalization lattice is divided into a set of (potentially overlapping) paths the global optimum can be determined by only comparing the local optima. Moreover, there is no need to evaluate the metrics for nodes that have been tagged predictively because they can never be a local or global optimum.

Metrics

In this section we briefly cover the most important monotonic *metrics*. The interested reader is referred to [EDI+09, DW99, LDR05, Sam01, Swe97, BA05] for further details. The *Height Metric* is utilized by the algorithm presented in [Sam01]. Similar to the *Precision Metric* [Swe02], it measures information loss solely based upon a state itself (i.e., its generalization levels) and is therefore independent of the actual input dataset. In [EDI+09] a monotonic version of the *Discernibility Metric* [BA05] has been presented, which estimates information loss based on the equivalence classes induced by a transformation. The *Entropy Metric* [DW99] compares the original input dataset with the transformed representation.

In the following sections we quickly review the two major previous algorithms that are relevant in our context. The algorithm of Samarati [Sam01] is an early optimal k-anonymity algorithm. Because it is only able to find an optimal solution for a very limited metric (*Height*) we will not discuss it in detail. Incognito implements a *horizontal traversal strategy* (i.e., traversing the lattice level by level), whereas OLA implements a *vertical traversal strategy* (i.e., jumping between levels).

The Incognito Algorithm

LeFevre et al. proposed the *Incognito* algorithm [LDR05], which implements an approach related to dynamic programming. The general idea is that if a transformed subset of the quasi identifiers is not k-anonymous, the transformation of the complete dataset can not be k-anonymous either. Therefore it constructs generalization lattices for each individual subset of n quasi identifiers and traverses them by performing a bottom-up, breadth-first search. It utilizes predictive tagging, to prune parts of the local search space. The states that have been found to be not anonymous in a subset of size $m < n$, can not be anonymous in a subset of size $m+1$. This allows to predictively tag states of the generalization lattices which are constructed in subsequent iterations. The algorithm halts when the lattice for all n quasi identifiers has been processed.

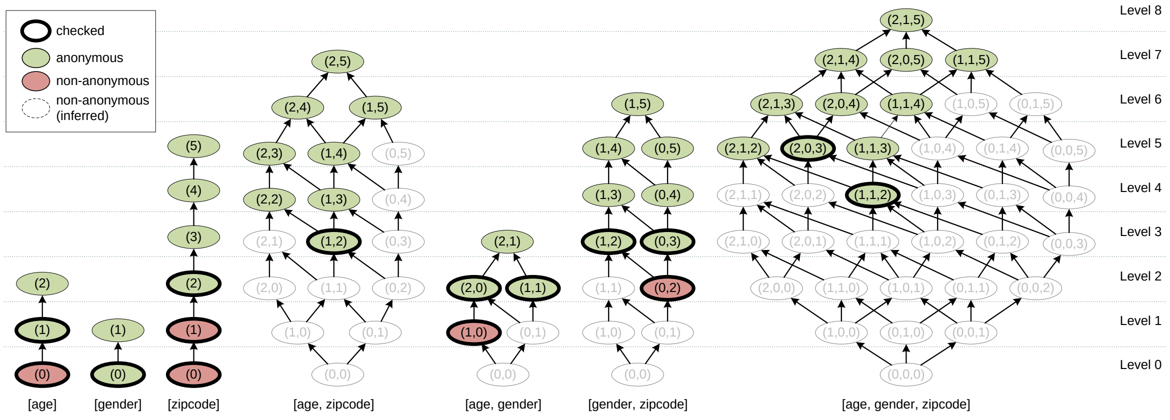


Figure 6.20: Example for the Incognito algorithm

An example is shown in Figure 6.20. It shows the lattices built by Incognito for the example dataset. The algorithm starts by focusing on the quasi identifier *age*. It builds a lattice and checks if the column is anonymous for the state (0). As this state is not k-anonymous, the algorithm proceeds with the state (1), which is k-anonymous. This results in predictive tagging of the state (2). The same procedure is applied to the quasi identifiers *gender* and *zipcode*. After checking all subsets of size one, Incognito proceeds with all subsets of size two. In this step it is possible to tag all nodes as non-anonymous that contain substates that were not anonymous in a previous iteration. E.g., in the lattice for the columns *age* and *zipcode* all nodes which define level 0 for *age* and level 0 or 1 for *zipcode* are tagged as being non-anonymous. Therefore the first state that is processed by Incognito in this lattice is the state $(1, 2)$. As this state represents a k-anonymous transformation, predictive tagging can be applied to all other nodes in this lattice. The other lattices are processed analogously [LDR05].

The OLA Algorithm

El Emam et al. proposed a k-anonymization algorithm, called *Optimal Lattice Anonymization* (OLA) [EDI+09] and showed that it outperforms the approaches presented in [Sam01] and [LDR05]. It implements a divide & conquer approach. The idea is to decompose a lattice into smaller sublattices and utilize predictive tagging to prune parts of the search space. A sublattice (b, t) is defined by a bottom node b and a top node t and contains b and t as well as all nodes that are generalizations of b and specializations of t. OLA starts by processing the complete lattice. It then constructs sublattices by enumerating all nodes M on level $\lfloor \frac{1}{2}(b.level + t.level) \rfloor$ of the current lattice. If a node $m \in M$ has not been tagged already, it is checked for k-anonymity and predictive tagging is applied. If m is tagged as anonymous the algorithm proceeds with the lower sublattice (b, m), otherwise it proceeds with the upper sublattice (m, t). This process halts when all sublattices have been enumerated.

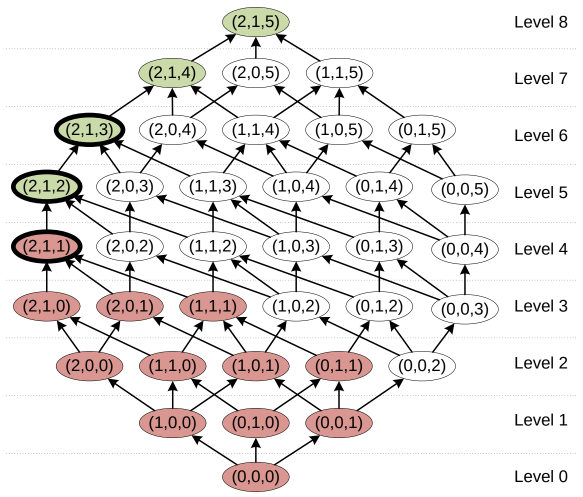


Figure 6.21: Example for the OLA algorithm

The first iteration of the algorithm for our example dataset is shown in Figure 6.21. It starts by enumerating all nodes on level 4 and we assume that it starts with node $(2, 1, 1)$. As this node has not been processed before, it is checked for anonymity. Because the state does not represent an anonymous transformation, $(2, 1, 1)$ as well as all of its specializations (dark gray) are tagged as non-anonymous. The algorithm then proceeds to the upper sublattice $((2, 1, 1), (2, 1, 5))$, which contains the light gray nodes. In the subsequent steps, it would construct the sublattices $((2, 1, 1), (2, 1, 3))$

and $((2,1,1),(2,1,2))$, effectively checking the nodes $(2,1,1)$, $(2,1,3)$ and $(2,1,2)$ to find the locally optimal two-anonymous node $(2,1,2)$. The algorithm would then proceed to the next node on level 4 which is $(2,0,2)$ and construct the according lower sublattice as this state is anonymous [EDI+09].

6.4.2 Implementation Framework

Our approach is based upon a generic framework for the efficient implementation of k-anonymity algorithms. This section presents the framework and describes the fundamental ideas behind it. Firstly, we assume that the process of checking individual states for k-anonymity is the main bottleneck for this class of anonymization algorithms and should be as efficient as possible. Secondly, we observe that general purpose database systems are not well suited for k-anonymity algorithms, because they have been designed for much more complex query and transaction processing. Thirdly, given the current trend towards main-memory data management as well as the applicability of data compression techniques, managing all data in main-memory is feasible.

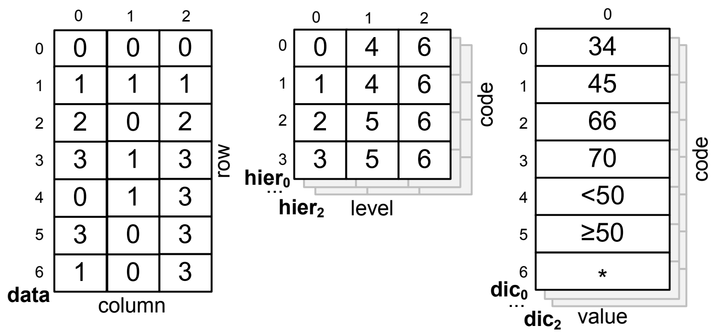


Figure 6.22: Example data representation

The groundwork of our framework is laid by a carefully designed memory layout, which enables the efficient application of different generalization strategies to an input dataset. Additionally, the anonymization operators are problem-aware and interwoven with the other parts of the algorithm. This allows for several further optimizations. The basic implementation including the first optimization can be used for all generalization-based anonymization algorithms which use global recoding. The other optimizations additionally require monotonic generalization hierarchies.

Basic implementation

In our framework, we hold all data in main memory and implement dictionary compression on all data items. Generalization hierarchies are represented as tables. A table representing the generalization hierarchy for the attribute *age* from Figure 6.18 is shown in Figure 6.8.

Level 0	Level 1	Level 2	Level 3
1	]1-25]	]1-50]	⋆
...	]1-25]	]1-50]	⋆
25	]1-25]	]1-50]	⋆
26	]25-50]	]1-50]	⋆
...	]25-50]	]1-50]	⋆
50	]25-50]	]1-50]	⋆
51	]50-75]	]50-100]	⋆
...	...	...	...
100	]75-100]	]50-100]	⋆

Table 6.8: Tabular generalization hierarchy

One dictionary $dic_0, ..., dic_{n-1}$ per quasi identifier is used to map the values contained in the corresponding column onto integer values. By encoding the values from the input dataset before the values contained in the higher levels of the generalization hierarchies it is guaranteed that the original values of a column with m distinct values get assigned the numbers 0 to $m-1$. This allows for an efficient representation of the generalization hierarchies $hier_0, ..., hier_{n-1}$ as two-dimensional arrays. An excerpt of the resulting memory layout for the example dataset is shown in Figure 6.22. The values of the attribute *age* from column 0 and the relevant values from the generalization hierarchy are encoded in the corresponding dictionary dic_0.

The associated generalization hierarchy $hier_0$ is represented as a two dimensional array in which the i-th row contains the values for the original data item which is encoded as i in the dictionary. The j-th column stores the corresponding transformed value at the j-th level of the hierarchy. The other quasi identifiers are handled analogously. The input dataset itself is represented as a row-oriented integer array ($data$). Additionally an equal data structure *buffer* is maintained which is used to store a transformed representation of the original data. Based on this memory layout, transforming a value from the input data in cell (row, col) to the value defined on level *level* of its generalization hierarchy and storing it in the buffer is a simple assignment:

$$buffer[row, col] \leftarrow hier_{col}[data[row, col], level] \tag{6.1}$$

When checking a state, the algorithm iterates over all rows in the dataset and applies the assignment (1) to each cell. Afterwards, the transformed row is passed to a groupify operator, which computes the equivalence classes by adding the rows to a hash table. Finally, the k-anonymity check is applied by checking whether all classes are of size ≥k. Moreover, a suppression parameter s can be specified. It defines an upper bound for the number of rows that can be suppressed in order to still consider a dataset k-anonymous. This further reduces information loss, as the minimum size k is not strictly enforced for all equivalence classes. Instead, classes of size $< k$ are removed from the dataset as long as the total number of suppressed rows is lower than a predefined threshold.

The presented basic implementation is already very efficient. As can be seen in the following complexity analysis, it has an amortized runtime complexity of $\mathcal{O}(n * m)$, where m is the number of rows and n is the number of columns.

- **Step 1:** The transformation of a single row from the input dataset is of complexity $\mathcal{O}(1)$ as it is only an assignment including indirections. This adds up to a complexity of $n * \mathcal{O}(1) = \mathcal{O}(n)$ for transforming the complete dataset.

- **Step 2:** Building the equivalence classes is implemented by sequentially adding all tuples to a hash table and updating an associated counter. Adding an element to a hash table has an amortized run-time complexity of $\mathcal{O}(1)$. Therefore building the equivalence classes for the complete dataset has an amortized run-time complexity of $n * \mathcal{O}(1) = \mathcal{O}(n)$.

- **Step 3:** Testing whether the equivalence classes fulfil the k-anonymity criterion is implemented by iterating over all entries in the hash table which has a worst-case complexity of $\mathcal{O}(n)$.

The amortized runtime complexity of checking where a given transformation is k-anonymous for a dataset with n rows is therefore $\mathcal{O}(n) + \mathcal{O}(n) + \mathcal{O}(n) = \mathcal{O}(n)$. The run-time complexity does not only depend on the number of rows n but also on the number of columns m. As the number of columns influences the run-time complexity of step 1 and step 2 by a linear factor of m the overall amortized runtime complexity depending on n and m is $\mathcal{O}(n * m)$.

Optimizations

The following section presents some additional optimizations that exploit similarities between states, which are checked consecutively by an algorithm.

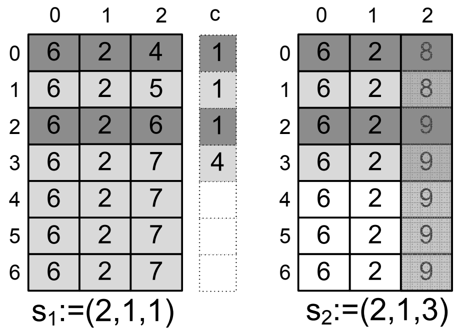


Figure 6.23: Roll-up and Projection

Projection: Because the transformed data is materialized in a buffer, it is possible to only transform these parts of the data that actually change. A *projection* can be applied if two consecutive states s_1 and s_2 define the same level of generalization for some quasi-identifiers. These columns are already represented in the correct state in the buffer and do not need to be transformed again. An example is shown in Figure 6.23. Here, only the column 2 needs to be transformed when checking the states $(2, 1, 1)$ and $(2, 1, 3)$. It is important to notice that an efficient implementation of this optimization can not just iterate over the set of active columns when processing each row. The overhead induced by this additional indirection generally exceeds the positive effect of not having to transform some cells. The implementation is therefore based on a branch table, which redirects the control flow to a parametrized implementation for each possible number of active columns. E.g., the implementation that transforms two columns takes the arguments $col1$ and $col2$ and executes the following for each row in the dataset:

$$buffer[row, col1] \leftarrow hier_{col1}[data[row, col1], level_{col1}]$$
$$buffer[row, col2] \leftarrow hier_{col2}[data[row, col2], level_{col2}]$$

Roll-Up: When an algorithm moves from a state s_1 to a state s_2 which is a generalization of s_1, the equivalence classes for s_2 can be built by merging the equivalence classes of s_1 if the generalization hierarchies are monotonic. This is called *roll-up* and our implementation is very efficient because we do not operate on top of a database (which is a black box) but are able to interweave the groupify operator with the anonymization algorithm. To this end we store the index of a representative (the first row that has been added for an equivalence class) in each entry in the hash table and manage a second groupify operator. We then iterate over the hash table entries that have resulted from the previous check and transform only the representative rows. Because we use a pointer into the original dataset instead of the actual equivalence classes we only introduce a very small space overhead and are again able to apply the assignment (1) instead of an actual join. The representative rows are then passed to the second groupify operator together with the size of their original classes. The operators are swapped (references are exchanged) prior to each check. A roll-up is also possible for the transition shown in Figure 6.23. In this figure, the classes are denoted by different shades of gray. We assume that the previous check resulted in the classes {0}, {1}, {2} and {3,4,5,6}. The representatives and sizes (r, c) are (0,1), (1,1), (2,1) and (3,4) as indicated by the additional rightmost column. To compute the classes for s_2 it is only necessary to transform and group four rows, whereas seven rows had to be processed to check the state s_1.

Taking snapshots: A series of roll-up operations can only be performed on a path of non-anonymous nodes. A similar technique can be applied in other state transitions, if a buffer is managed which contains snapshots of the equivalence classes of previous states (called *history*). The equivalence classes for a state s can then be built by merging the classes defined by a more specialized state s'. If multiple suitable snapshots are available, we pick the one that contains the fewest classes. Due to the presence of pointers to representative rows in our groupify operator we are able to only store references to the representative and the size for each equivalence class. This compact representation (8 bytes per class) allows us to efficiently maintain a large number of snapshots. Furthermore, it is only necessary to store snapshots for non-anonymous nodes, because otherwise all generalizations will be tagged predictively. A snapshot can also be pruned if all nodes that could possibly utilize it are already tagged. A simple check that covers parts of this restriction is to look at all direct generalizations of the state and discard the snapshot if these are all tagged as k-anonymous.

To further limit the memory consumption and prevent performance degradation we introduce two parameters that control the buffer. A threshold defines the maximum number of equivalence classes (relative to the number of data items in the original dataset) a snapshot is allowed to contain in order to be stored. A second parameter defines an upper bound for the number of entries that can be stored in the buffer at the same time. When this limit is reached we first check all contained snapshots for whether they can be dropped by looking at their direct generalizations. Otherwise, the least recently used snapshot is dropped. Those parameters allow to tune the solution according to a space-time trade-off. The described buffering of snapshots does not fully replace the roll-up optimization but complements it, as it only stores a limited amount of snapshots of a predefined maximum size.

Putting it all together: The individual optimizations can be combined with each other. The transition from Figure 6.23 can, e.g., benefit from performing a roll-up as well as a projection. There are several limitations for valid sate changes that ensure that the data is always in a consistent state, though. For example, if a roll-up optimization was performed in the previous iteration but can not be applied in the current iteration, it is not possible to only transform a subset of the columns. A tabular representation of all valid state transitions is shown in Table 6.9.

The possible combinations of the described optimizations are denoted by a combination of labels: *AR* $\hat{=}$ *All Rows*, *RU* $\hat{=}$ *Roll-Up*, *SS* $\hat{=}$ *Snapshot*, *AC* $\hat{=}$ *All Columns* and *CC* $\hat{=}$ *Changed Columns*. Moreover, state transitions depend on the relationship between the previous state s and the current state s' which we denote by different

	AR, AC	AR, CC	RU, AC	RU, CC	SS, AC	SS, CC
AR, AC	$\prec$	$\preceq$	$\succ$	$\succeq$	$\exists \succ$	$\exists \succeq$
AR, CC	$\prec$	$\preceq$	$\succ$	$\succeq$	$\exists \succ$	$\exists \succeq$
RU, AC	$\prec$,$\preceq$	-	$\succ$	$\succeq$	$\exists \succ$,$\exists \succeq$	-
RU, CC	$\prec$,$\preceq$	-	$\succ$	$\succeq$	$\exists \succ$,$\exists \succeq$	-
SS, AC	$\prec$,$\preceq$	-	$\succ$	$\succeq$	$\exists \succ$,$\exists \succeq$	-
SS, CC	$\prec$,$\preceq$	-	$\succ$	$\succeq$	$\exists \succ$,$\exists \succeq$	-

Table 6.9: Possible transitions

symbols in Table 6.9. $s' \succ s$ denotes that s' is a generalization of s, $s' \prec s$ denotes that s' is a specialization of s, $s' \succeq s$ implies $s' \succ s$ and $s' \preceq s$ implies $s' \prec s$ but with at least one equal generalization level for any quasi identifier. $\exists_s : s' \succ s$ and $\exists_s : s' \succeq s$ denotes that a snapshot s with $s' \succ s$ or $s' \succeq s$ is stored in the buffer. Despite the existence of snapshots, the relationships are mutually exclusive. If it is possible to perform a roll-up or utilize a snapshot, we always perform a roll-up. Otherwise utilizing a snapshot is always preferred over other implementations. As can be seen in Table 6.9, state transitions are mainly limited in case of projections. Projections can only be performed if the current transformation implements a roll-up or all rows have been transformed in the previous state.

6.4.3 The FLASH Algorithm

In this section we present our novel FLASH algorithm. It traverses the lattice in a bottom-up breadth-first manner and constantly generates paths, which ramify like lightning flashes. It is based upon the following observations. Firstly, predictive tagging can be exploited best if the lattice is traversed vertically and in a binary fashion. Secondly, when traversing a lattice vertically, the execution time becomes volatile regarding the representation of the input dataset (e.g., the order of the columns). This has to be prevented by implementing a stable strategy. Thirdly, in order to achieve the best performance, the algorithm should prefer nodes that allow to apply the previously presented optimizations.

Basic Algorithm

As shown in Algorithm 7, FLASH iterates over all levels in the lattice. It enumerates all nodes on each level and calls FindPath(*node*) if it is not tagged. As shown in Algorithm 8, this function implements a greedy depth-first search which terminates at the top node or when a node does not have an untagged successor.

When a path has been built, the function CheckPath(*path*, *pq*) is called. As can be seen in Algorithm 9, it implements a binary-search. It starts by checking the node at position $\lfloor \frac{1}{2}(path.size - 1) \rfloor$. Whenever a node is checked for k-anonymity, we also apply predictive tagging within the whole generalization lattice. Depending on the result of the check, the algorithm then proceeds with the lower or upper half of the path. Whenever a node is explicitly checked and determined to be non-anonymous, we add it to a priority queue (*pq*). If a node is explicitly checked and determined to be anonymous, we store a reference to it, as it could be the local optimum. There is no need to check whether a node has already been tagged, because by definition Algorithm 8 always returns a path of untagged nodes. Additionally, predictive tagging is always applied in the direction opposite to the one taken by the algorithm. Another important thing to notice is that after the search terminates the variable *optimum* will always hold a reference to the optimal anonymous node on the path (if there is any).

Algorithm 7: Outer loop of the FLASH algorithm

```
Input: Lattice $lattice$
begin
    $pq \leftarrow$ $new\ priority\ queue$
    for $l = 0 \rightarrow lattice.height - 1$ do
        foreach $node \in level[l]$ do
            if $!node.tagged$ then
                $path \leftarrow$ FINDPATH($node$)
                CHECKPATH($path, pq$)
                while $!pq.isEmpty$ do
                    $node \leftarrow pq.extractMin$
                    foreach $up \in node.successors$ do
                        if $!up.tagged$ then
                            $path \leftarrow$ FINDPATH($up$)
                            CHECKPATH($path, pq$)
```

The globally optimal node is determined by comparing the current local optimum with the current global optimum in STORE(*optimum*).

After a path has been checked, the nodes in the priority queue are utilized to build new paths starting from the successors of the non-anonymous nodes that have been explicitly checked when processing the previous path. The general idea behind this is that these paths increases the potential to apply the roll-up and snapshot optimizations. Somewhat simplified (see Section 9), the priority queue returns the nodes according to their level in the lattice. We start with the minimal element as this potentially increases the length of the created path. This in turn increases the potential to apply predictive tagging to other nodes in the lattice.

In case of an empty priority queue, the algorithm proceeds with the outer loop. It terminates when the outer loop terminates. As explained in the following section, our algorithm further implements a general strategy that induces a total order on all nodes in the lattice. Whenever we iterate over a set of nodes we follow this order. This includes all steps which are highlighted in gray in the presented pseudocodes.

Traversal Strategy

An important property of an algorithm that implements a vertical traversal strategy is stability in terms of execution time. This is due to the fact that the order in which nodes are enumerated influences the order in which the nodes are checked. In combination with a vertical strategy and predictive tagging this leads to different lattice traversals, which leads to a varying number of k-anonymity checks. This finally leads to differences in the algorithms' execution times. In practice these differences can, e.g., be triggered by changing the order of the columns in the input dataset (see Section 6.4.4). The solution to this problem is to apply a fixed strategy whenever nodes are enumerated. The idea behind our strategy is to prefer nodes with a lower degree of generalization.

A node is a tuple $n = (n_0, ..., n_j)$ with $0 \leq n_i \leq m_i$ for all $0 \leq i \leq j$, where m_i defines the maximum level in the hierarchy of the i-th quasi identifier. Moreover, $distinct(i, l) = |\{hier_i[x][y] \mid y = l\}|$ returns the distinct number of values on level l

Algorithm 8: FINDPATH(NODE)

```
Input: Start node $node$
Result: Path of untagged nodes $path$
begin
    $path \leftarrow new\ list$
    while $path.head() \neq node$ do
        $path.add(node)$
        foreach $up \in node.successors$ do
            if $!up.tagged$ then
                $node \leftarrow up$
                break
    return $path$
```

Algorithm 9: CHECKPATH(PATH, PQ)

```
Input: Path $path$, priority queue $pq$
begin
    $low \leftarrow 0$; $high \leftarrow path.size - 1$; $optimum \leftarrow null$
    while $low \leq high$ do
        $mid \leftarrow \lfloor \frac{1}{2}(low + high) \rfloor$
        $node \leftarrow path.get(mid)$
        if CHECKANDTAG(NODE) then
            $optimum \leftarrow node$; $high = mid - 1$
        else
            $pq.add(node)$; $low = mid + 1$
    STORE($optimum$)
```

of the i-th generalization hierarchy. We define three criteria for each node n. The criterion $c_1(n)$ returns the height of the node in the lattice:

$$c_1(n) = \sum_{i=0}^{j} n_i \tag{6.2}$$

The criterion $c_2(n)$ is used to differentiate between nodes on the same level and resembles the Precision metric. It returns the average generalization over all quasi identifiers:

$$c_2(n) = \sum_{i=0}^{j} \frac{n_i}{m_i} \tag{6.3}$$

Finally, the criterion $c_3(n)$ is utilized to differentiate between nodes on the same level which also describe the same average generalization. It represents the average number of distinct values on the current level of each quasi identifier:

$$c_3(n) = 1 - \sum_{i=0}^{j} \frac{distinct(i, n_i)}{distinct(i, 0)} \tag{6.4}$$

These three criteria are then combined into a vector in $\mathbb{R}^3$:

$$c(n) = \begin{pmatrix} c_1(n) \\ c_2(n) \\ c_3(n) \end{pmatrix} \tag{6.5}$$

The nodes are traversed according to the totally ordered vector space induced by the lexicographical order, i.e. $c(n_1) \leq c(n_2)$ iff:

- $c_1(n_1) < c_1(n_2)$, or
- $c_1(n_1) = c_1(n_2) \wedge c_2(n_1) < c_2(n_2)$, or
- $c_1(n_1) = c_1(n_2) \wedge c_2(n_1) = c_2(n_2) \wedge c_3(n_1) \leq c_3(n_2)$.

We apply this order when enumerating nodes on a level in the outer loop and when enumerating successors of a node in FindPath. Finally, the priority function also serves as the key for the entries in the priority queue. The strategy has to be implemented carefully. Sorting all levels in the lattice and all pointers to successors prior to the execution of the algorithm is too expensive. We therefore evaluate the priority function lazily and sort a nodes' successors only when needed. Analogously, a level is sorted directly before iterating over it. This allows to exclude all nodes from sorting which are already tagged.

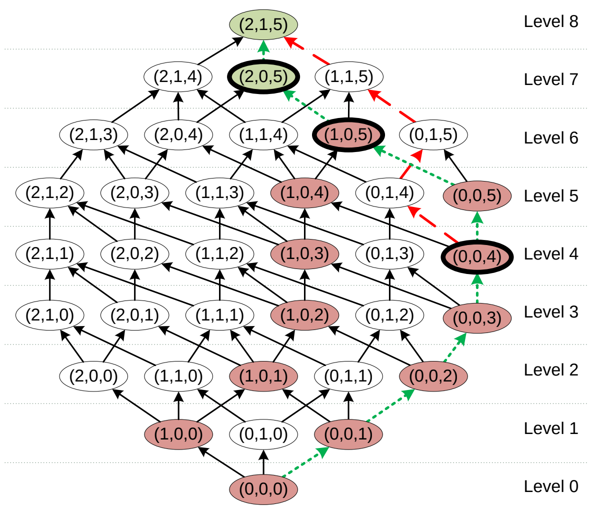


Figure 6.24: Example for the FLASH algorithm

Example

The first two iterations of the FLASH algorithm can be seen in Figure 6.24. It starts by building a path from the bottom node $(0,0,0)$ to the top node $(2,1,5)$. This results in the rightmost path that is indicated by the dotted lines. This path is then checked in a binary-manner which results in explicit checks of the nodes $(0,0,4)$, $(1,0,5)$ (non-anonymous) and $(2,0,5)$ (anonymous). The states of the other non-anonymous nodes (dark gray) are determined by applying predictive tagging from $(0,0,4)$ and $(1,0,5)$. The top-node is predictively tagged as being anonymous from the node $(2,0,5)$. After checking the first path, the priority queue contains the nodes $(1,0,5)$ and $(0,0,4)$. As

$(0, 0, 4)$ is prioritized according to our general strategy, we start to build an new path from $(0, 0, 4)$, which is denoted by the dashed lines. When the priority queue is empty, the algorithm builds the next path starting from $(2, 0, 0)$ because $(0, 1, 0)$ has already been tagged.

Dataset	QIs	Records	States	Size [MB]	Init [ms]
ADULT [1]	9	30,162	12,960	2.52	86
CUP [2]	8	63,441	45,000	7.11	334
FARS [3]	8	100,937	20,736	7.19	152
ATUS [4]	9	539,253	34,992	84.03	1,031
IHIS [5]	9	1,193,504	25,920	107.56	1,627

Table 6.10: Evaluation datasets

6.4.4 Evaluation

In the experiments we used five real-world datasets, most of which have already been utilized for benchmarking previous work on k-anonymity. We evaluate the execution times and memory consumption of our algorithm and compare it with previous work.

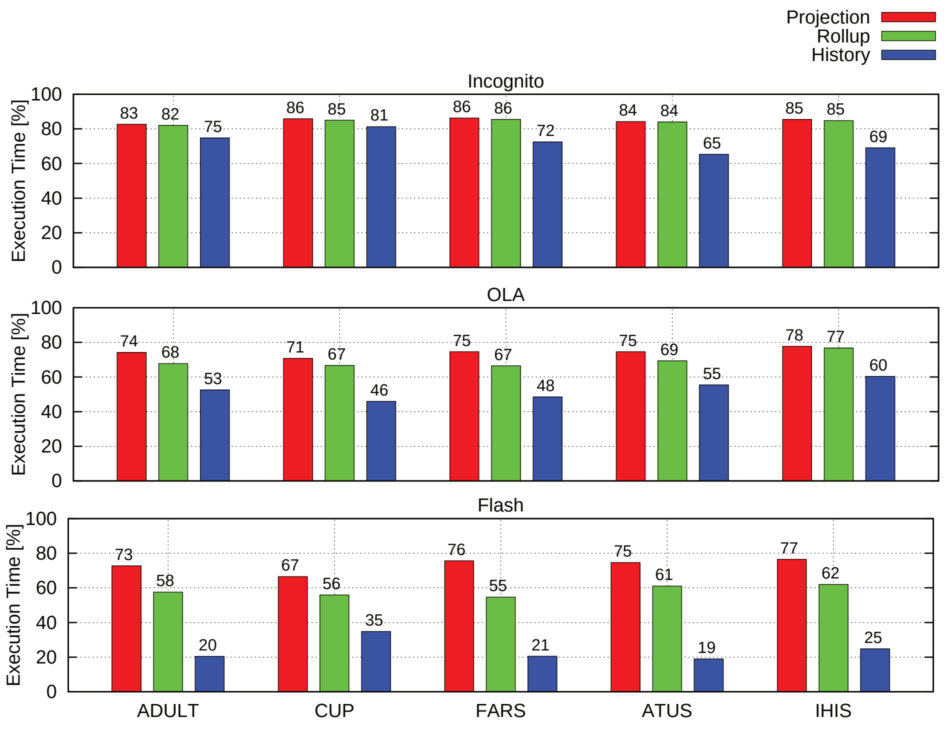


Figure 6.25: Effectiveness of optimization levels

Datasets

The datasets include the 1994 US census database (ADULT), KDD Cup 1998 data (CUP), NHTSA crash statistics (FARS), the American Time Use Survey (ATUS) and the Integrated Health Interview Series (IHIS). The ADULT dataset serves as a de-facto standard for the evaluation of k-anonymity algorithms. An overview over the datasets is shown in Figure 6.10. They cover a wide spectrum, ranging from about

[1]http://archive.ics.uci.edu/ml/datasets/adult
[2]http://kdd.ics.uci.edu/databases/kddcup98/kddcup98.html
[3]http://www-fars.nhtsa.dot.gov/main/index.aspx
[4]http://atusdata.org/index.shtml
[5]http://www.ihis.us/

30k to 1.2M rows (2.52 MB to 107.56 MB) consisting of eight or nine quasi identifiers. The associated generalization hierarchies have a height between 1 and 5 levels. The number of states in the generalization lattice, which is defined by the number of quasi identifiers as well as the height of the associated hierarchies, ranges from 12,960 for the ADULT dataset to 45,000 for the CUP dataset.

Setup

The benchmarks were performed on a Desktop machine equipped with a 4-core 3.1 GHz Intel Core i5 CPU running a 64-bit Linux 3.0.14 kernel. The algorithm was implemented in Java and executed on a 64-bit Sun JVM (1.7.0) with a heap size of 512 MB. We anonymized each of the datasets with $2 \leq k \leq 10$, suppression rates s of 0%, 2% and 4% and the monotonic *Discernibility Metric (DM*)*. We incrementally enabled the optimizations, resulting in four different configurations. We executed each of these configurations five times for each algorithm (Incognito, OLA and FLASH) and combination of the parameters k and s, which resulted in 135 runs per dataset. The threshold for the maximum size of a snapshot was set to 20% and the size of the history was limited to 200 entries. The results are reported without the time needed for initialization, which includes reading the entire data from disk and performing dictionary encoding (see Figure 6.10). The execution time of all three algorithms is dominated by the time spent on k-anonymity checks. As these are implemented analogously for all algorithms, the comparison is fair.

Overview

Figure 6.25 shows the workload averages (geometric mean over all values of k and s) for each dataset and algorithm. The basic implementation, which does not implement any optimizations, defines the 100% baseline. The optimizations are enabled incrementally and it can be seen, that all optimizations have a positive effect on the execution times of all algorithms for all datasets. It is important to note that none of the optimizations yields any overhead for any configuration. The FLASH algorithm benefits the most, as it was explicitly designed to fully exploit the framework. Incognito benefits the least, because of its horizontal traversal strategy, which, e.g., completely prevents the roll-up optimization. Figure 6.26 presents a comparison of the algorithms in terms of the geometric mean (logarithmic scale) of the execution times in seconds. FLASH outperforms all algorithms and the speedup compared to the state-of-the-art (OLA) ranges from about 37% for CUP to a factor of almost 3 for IHIS.

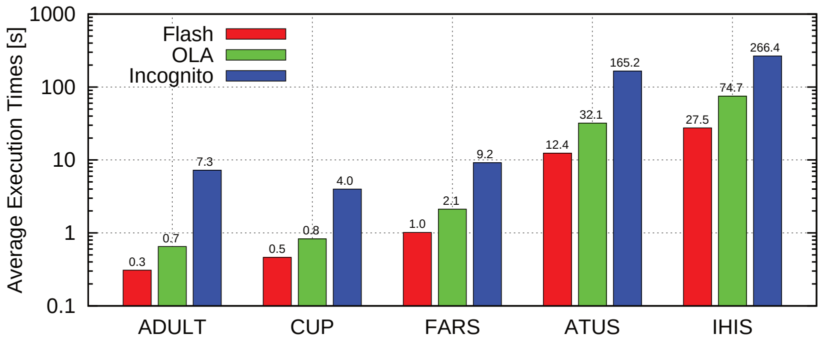


Figure 6.26: Comparison of average execution times

Details

Figure 6.27 shows a comparison of the actual execution times of the algorithms for $2 \leq k \leq 10$ on all configurations and datasets. It can be seen that the execution times of the algorithms do not only differ by a constant factor, but follow different trends. In the context of an individual configuration, FLASH offers an almost constant execution time and outperforms the other algorithms for each value of k. The results are summarized in Tables 6.11 and 6.12. Table 6.11 shows the minimum and the maximum of the execution times of FLASH for $2 \leq k \leq 10$ on all datasets and suppression rates in seconds. It can be seen that FLASH is able to find the optimal solution in well under 4 seconds for all configurations despite ATUS and IHIS with 2% and 4% suppression rates. In these cases, more checks have to be performed to find the optimal solution, which leads to execution times in the order of a minute.

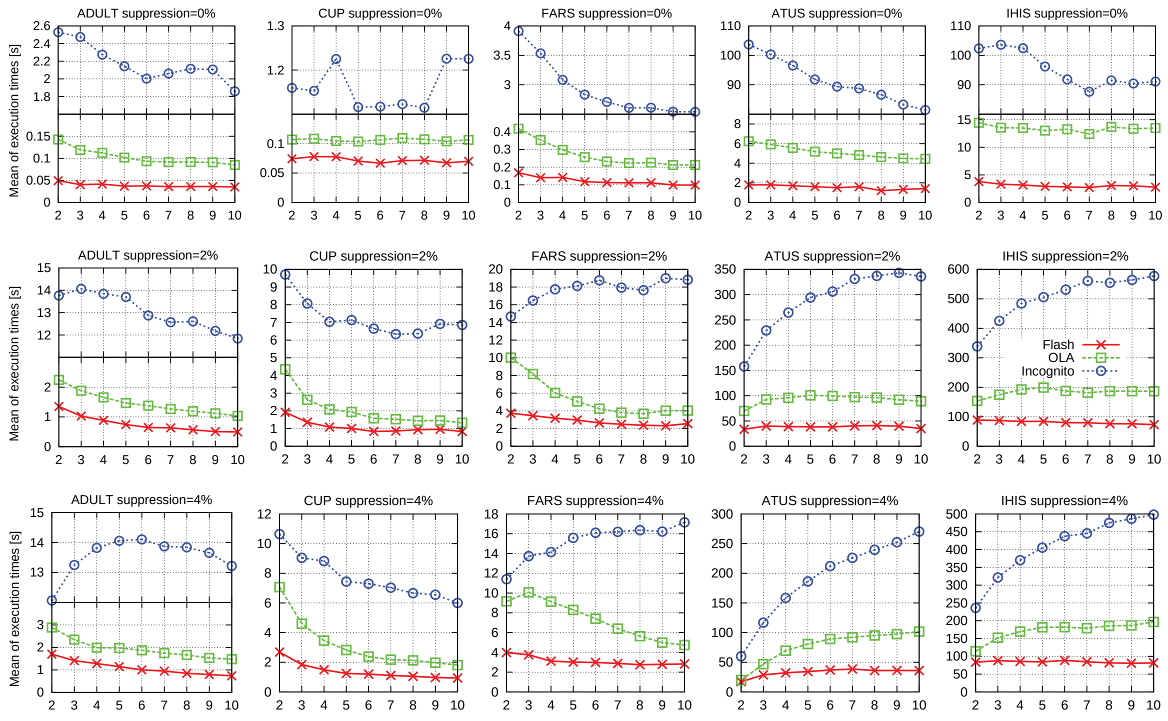


Figure 6.27: Execution times for 2 ≤ k ≤ 10

Table 6.12 compares the performance of Incognito and OLA to our algorithm. It shows the minimum and maximum factors between the execution times of the other algorithms and FLASH for $2 \leq k \leq 10$ and each configuration. It can be seen, that FLASH outperforms the other algorithms for all configurations on all datasets. The largest performance gain in comparison to the Incognito algorithm exists for a 0% suppression rate with a factor of more than 70 for ATUS ($k = 8$). FLASH especially outperforms OLA on the two largest datasets at a 0% suppression rate, with a factor of about 4 for ATUS ($k = 8$) and a factor of about 5 for IHIS ($k = 10$).

	ADULT	CUP	FARS	ATUS	IHIS
s=0%	0.04 – 0.05	0.07 – 0.08	0.10 – 0.17	1.20 – 1.80	2.72 – 3.79
s=2%	0.50 – 1.35	0.83 – 1.92	2.31 – 3.74	33.55 – 40.95	73.22 – 89.04
s=4%	0.74 – 1.70	0.94 – 2.69	2.75 – 3.99	17.11 – 38.65	80.51 – 88.63

Table 6.11: Execution times of the FLASH algorithm [s]

The experiments also show that our generic framework is suitable for the efficient implementation of other algorithms for optimal k-anonymity. Furthermore, OLA clearly outperforms the Incognito algorithm, even up to an order of magnitude in some cases. We have not implemented the *SuperRoots* variant of Incognito, because our experiments revealed that it is only beneficial without suppression and decreases the performance in all other cases. FLASH outperforms OLA in all cases, although our implementation of OLA is already highly efficient and includes several optimizations [KPE+12b].

	ADULT	CUP	FARS	ATUS	IHIS
	s=0%				
OLA	2.4 – 3.0	1.3 – 1.6	2.0 – 2.5	3.0 – 3.9	3.8 – 4.9
Incognito	51.2 – 61.3	14.8 – 18.2	21.8 – 25.8	55.4 – 72.2	27.0 – 32.9
	s=2%				
OLA	1.7 – 2.2	1.5 – 2.3	1.5 – 2.7	2.1 – 2.6	1.7 – 2.6
Incognito	10.2 – 24.0	5.1 – 8.2	3.9 – 8.2	4.7 – 9.6	3.8 – 7.9
	s=4%				
OLA	1.6 – 2.0	1.9 – 2.6	1.7 – 2.9	1.2 – 2.8	1.4 – 2.4
Incognito	7.1 – 17.9	4.0 – 6.7	2.9 – 6.1	3.5 – 7.5	2.8 – 6.1

Table 6.12: Performance of Incognito and OLA compared to FLASH [factor]

Table 6.13 presents a comparison of the algorithms' memory consumption. The snapshots stored in the history dominate the overall memory footprint. Incognito uses the least memory, whereas OLA uses the most. Incognito has the lowest memory usage, because many nodes in the largest generalization lattice (for all quasi identifiers) are already tagged when the lattice is traversed. It therefore creates significantly fewer snapshots than the other algorithms. FLASH is designed to immediately exploit snapshots, which leads to early eviction of many entries in the buffer and a memory consumption between Incognito and OLA.

Dataset	FLASH	OLA	Incognito
ADULT	8 – 15	10 – 18	8 – 13
CUP	39 – 55	44 – 79	37 – 43
FARS	20 – 46	22 – 55	20 – 37
ATUS	81 – 155	87 – 163	80 – 146
IHIS	166 – 432	172 – 471	162 – 376

Table 6.13: Memory consumption [MB]

Although the FLASH algorithm has been designed to leverage our implementation framework as much as possible, it also offers competitive performance in other implementation scenarios. This can be seen by comparing FLASH and OLA when all optimizations are disabled and the execution times are basically defined by the number of k-anonymity checks, which are performed by the algorithms. In this case, FLASH is up to 30% slower for the ADULT and FARS dataset with a 0% suppression rate. As this are the cases in which the execution times are very low, FLASH still provides very good performance, though. In all other cases (including the larger datasets), the performance of FLASH and OLA differs only by up to 10% and there is no clear winner. As soon as the optimizations are enabled, FLASH outperforms OLA in all configurations by large margins. Moreover, FLASH offers algorithmic stability as is described in the following section.

Algorithmic Stability

In contrast to Incognito, FLASH and OLA implement a vertical traversal strategy but only FLASH offers a stable execution time. Stable means that, the execution time of the algorithm does not depend on the order of the columns in the input dataset or the algorithm that is used to build the generalization lattice. E.g., a node $(1, 0, 0)$ represents a different transformation if the first column is swapped with another column in the dataset. This is an important property, because otherwise the behaviour of an algorithm is not reproducible without an explicit definition of these surrounding conditions.

As this property has not been discussed in any previous work, we assume that the experiments therein have been performed based on a *natural ordering*. Firstly, the order of the columns in the input datasets has been preserved as is. Secondly, the successors of a node are ordered according to which quasi identifier has been incremented. Thirdly, the nodes on a level are enumerated in the same way, assuming a breadth-first strategy during the lattice building process. This strategy has also been used for all example lattices in this section and in our experiments (see, e.g., Figure 6.19). We were able to closely reproduce the previously published results.

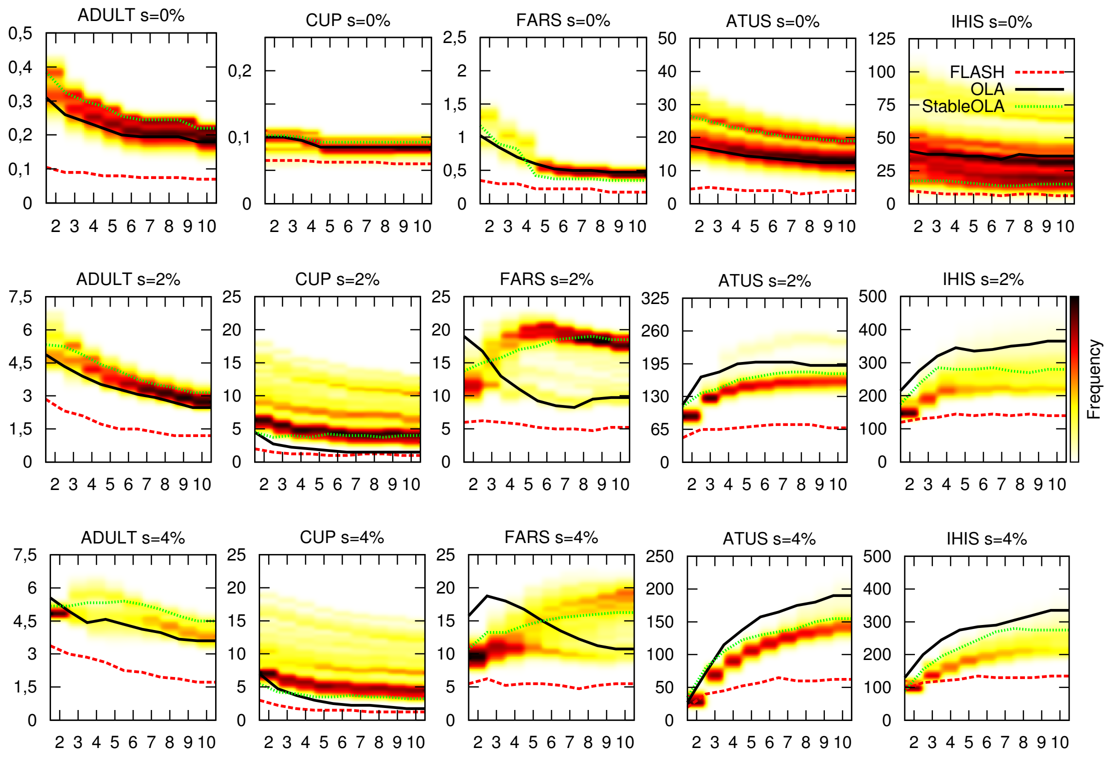


Figure 6.28: Distribution of the execution times of OLA for $2 \leq k \leq 10$

The distribution of OLA's execution times can be determined, if a fixed lattice building algorithm is used and OLA is executed for all permutations of the columns in the input datasets. Unfortunately this is not feasible, as the number of permutations is factorial in the number of columns (e.g. $9! = 362, 880$). Instead, we precomputed the anonymity property for all states in the lattices and used this information to simulate the execution of OLA for all permutations. The simulations take all optimizations from our framework into account. Each simulation results in a sequence of k-anonymity checks T, which were performed while processing one permutation. Each check $t \in T$

is defined by the number of active columns (t_c), as induced by a projection, and the number of active rows (t_r), as induced by the roll-up and history optimizations. The overall number of columns in the dataset is denoted by c. Based on this information we developed a cost model, which estimates the overall execution time. For one check $t \in T$, the costs for transforming the data are defined as $costs_t(t) = t_c \cdot t_r$, which is the number of cells that had to be transformed. In addition, the costs for grouping the data are $costs_g(t) = c \cdot t_r$, because the costs of grouping always depends on the total number of columns in the dataset. In order to fit the resulting costs to the actual times measured in the experiments, we divided the results by a constant factor. The overall costs for a simulation T were defined as:

$$costs(T) = \sum_{t \in T} costs_t(t) + costs_g(t) = \sum_{t \in T} t_c \cdot t_r + c \cdot t_r$$

Figure 6.28 shows the distribution of the costs for different values of k and selected datasets. The frequency of a cost estimate is represented by colors ranging from white (lowest) to black (highest). The dashed lines (OLA) represent the costs of the standard OLA algorithm with natural ordering, which has been used in the other experiments (e.g., in Figure 6.27). The dotted lines (StableOLA) show the costs of an implementation of OLA, which uses the same strategy as the FLASH algorithm and is thus stable. To this end, the nodes on level $\lfloor \frac{1}{2}(b.level + t.level) \rfloor$ for a sublattice (b, t) are enumerated in the order induced by our traversal strategy (see Section 9). The solid lines represent the costs of FLASH.

As can be seen when comparing Figure 6.28 to Figure 6.27, our cost model closely resembles the actual execution times of the algorithms. Despite for the FARS dataset at 2% and 4% suppression rates, the trends of the resulting frequencies are very similar to the trends measured in our experiments. In contrast, the costs estimated for StableOLA, always follows the overall trend. On average, this variant does not outperform standard OLA. The costs of OLA vary greatly, e.g., by up to a factor of 38 for IHIS with s=0%. The performance with natural ordering is sometimes excellent (e.g. CUP, s=4%) and sometimes poor (e.g. IHIS, s=2%). In contrast, the FLASH algorithm offers a stable execution time, which outperforms OLA with natural ordering and OLA with the same stable strategy in all cases. FLASH even outperforms the best run of OLA – the fastest permutation, which can not be predicted efficiently – in 114 out of 135 cases.

In the original work on OLA, the authors reported a total execution time of 20 s for the ADULT dataset ($k = 5$, $s = 1\%$) on a comparable testbed [Ema10]. With our framework, OLA was able to compute the optimal solution for the exact same configuration in 550 ms, whereas FLASH needed 310 ms (both numbers include initialization). In [LDR05] the authors also published results for the ADULT dataset ($k = 5$, $s = 0\%$) and an implementation of Incognito on top of a relational database system. In contrast to more than 3 minutes in the original work, our implementation of Incognito was able to solve this problem in less than 1s (including initialization) for the exact same configuration. FLASH needed about 100 ms. This shows that a large performance gain can be achieved by implementing a dedicated data management framework for k-anonymity algorithms.

6.5 Summary and Perspectives

In this chapter, several methods for browsing, editing and transforming RDF data have been proposed. These allow to enrich the dataspace with comprehensive annotations, to materialize implicit relationships between the data items and to anonymize sensitive datasets. We have provided extensive evaluations, which were utilized to choose the appropriate external tools for inclusion into our scripting environment and which provide important information for data integrators to estimate the complexity of common data integration tasks.

Future work on navigating RDF data could investigate how a prioritization of predicates can be utilized to provide a better overview over the currently displayed subgraph to the user. This means that the component would, e.g., prioritize predicates from the RDF/S schema definition vocabulary over instance-level predicates to increase the available context information.

User-defined data transformation scripts are executed by a dedicated workflow engine, which applies application-specific heuristics for distributing the load amongst the available computing nodes in order to achieve good performance. It is able to consider complex relationships and requires very little knowledge about the characteristics of the individual operations.

Possible directions for further improving the workflow engine include investigating how real-time load data could be utilized to prevent dispatching a script to a node which is currently under heavy load due to query processing. In the described configuration, the engine is also able to allocate scripts to nodes which do not host any of the required data sources. This is based upon the assumption that data access does not dominate the load profiles of most scripts. The engine could make more informed decisions if the volume of the required data would be known. In this context, cardinality estimations could be performed if information about the queries issued by the scripts would be available in advance, e.g., by requiring the user to explicitly annotate scripts with these metadata.

The querying interface and the scripting language are provided for extracting relevant data from the dataspace. In the context of semantic reasoning, future research could be oriented towards executing complex ontology segmentation or modularization algorithms in a distributed environment. These operators would potentially allow data integrators to much more easily extract and access relevant data.

The scripting environment offers a powerful data anonymization operator. This operator is based upon a generic implementation framework for globally-optimal full-domain k-anonymity and a novel k-anonymity algorithm. It clearly outperforms previous approaches and offers algorithmic stability by solely relying on a consistent strategy that implies a total order among the nodes in the generalization lattice. Even when it is not implemented on top of the presented framework, it offers a highly competitive performance. An additional feature of the framework is that, monotonic metrics can be evaluated with nearly no additional costs. As all locally optimal nodes are explicitly checked for k-anonymity, complex metrics, which are based on the data itself or the resulting equivalence classes (e.g, *Entropy*), can directly access the transformed data in the buffer or the equivalence classes in the hash table.

In this context, it would be interesting to investigate how the l-diversity and t-closeness properties could be integrated into the anonymization framework. As these properties rely on building frequency sets over a set of predefined sensitive attributes, a major challenge consists in integrating this process into the proposed optimizations. Future work could also try to better leverage the capabilities of modern multi-core processors by parallelizing the implementation framework and the FLASH algorithm. In case of limited availability of main memory or very large datasets, a disk-based implementation of a k-anonymity algorithm might be needed. Although this can be implemented on top of a relational database system, future work could investigate how the presented optimizations can be integrated into a disk-based version of the framework. The integration solution presented in this thesis is based upon a distributed graph structure. It would therefore be interesting to investigate to which extend distributed k-anonymity algorithms and algorithms for anonymizing graph-structured data could be applied in our context.

The scripting environment could also be extended by providing further operators. For example, the *Silk Link Discovery Framework* implements a rule-based system for discovering relationships between data items in the Web of Data [JIB10]. Here, the links are specified in an XML vocabulary called *Link Specification Language* (LSL). Similar to the approach presented in this thesis, the SILK framework is able to access several RDF datasets via a SPARQL interface. It then automatically performs record linkage and duplicate detection based on the link specification provided by the user. It provides different metrics which can be used for comparing data items and also allows the user to provide rules that split the data into different blocks in order to avoid processing the complete cartesian product [IJB11]. In our context, the Silk framework would be interesting as a component for performing record linkage and entity resolution. In order to decide which operators could be useful, a concrete use-case with a clear definition of the requirements would be required, though.

CHAPTER 7

Putting It All Together: A Prototypical Implementation

In the previous sections, individual components have been proposed for building an incremental ontology-based integration solution for translational medical research. This section presents a prototypical implementation called *VIOLIN* (**V**irtual **I**ncremental **O**nto**L**ogy-Based **IN**tegration). It integrates the presented components into a comprehensive integration system, which utilizes a tailored network protocol. The system further provides a graphical user interface (GUI) called VIOLIN Workbench. The workbench is oriented towards data integrators and supports data integration and maintenance tasks. Additionally, the querying interface, which is exhaustively utilized by the internal components of the system itself, is exposed to users and applications.

7.1 Implementation Details

In this section we describe the most important implementation details of our prototype. We start by giving an overview over the data architecture, which encapsulates the individual components into primitives that can be altered by data integrators to manage the dataspace. We then cover the underlying permission model, describe the network protocol and give a detailed description of the overall system architecture of the prototype.

7.1.1 Data Architecture

A UML class diagram covering the basic data architecture is shown in Figure 7.1. The individual primitives are structured in an hierarchical manner. As the overall system is inherently distributed, the most important primitive is a *Site*. A site represents a specific node within the system, which is reachable via a dedicated network endpoint. Within the application domain, a site is also the major primitive for the separation of data in terms of organizational structures. To this end, each site can represent a specific institution, department or research project, which is completely under local control. Conceptually, the central node itself (*Global*) is also an instance of a site, which is the parent of all other sites. Each site is allowed to export an arbitrary number of RDF graphs (*Sources*). A *Schema* is a specification of a source, which does only contain high-level metadata. If a schema exists, it is directly related to one of the site's sources. A schema can only exist for non-native RDF datasets, which have been created by an *Extraction* process. An extraction is utilized to derive an RDF-representation of a non-RDF dataset. Within the overall system, this is implemented for *Relational* databases as well as *HL7* message streams (see Chapter 4). A *Transformation* is another important architectural primitive, which represents a data transformation script (see

Chapter 6). Each primitive is an instance of the class *Element*. Each element has a unique identifier which is utilized to implement a fine-grained permission model, as will be described in the following section.

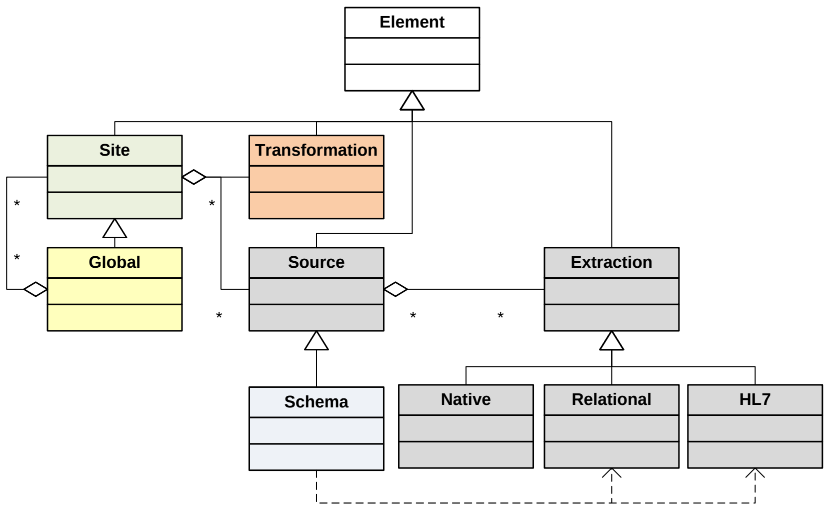


Figure 7.1: UML class diagram of the data architecture

7.1.2 Permission Model

The permission model consists of roles, which encapsulate a set of basic access rights for a set of primitives from the data architecture. With the exception of data transformation scripts, this light-weight concept offers a fine-grained mechanism to control the access rights of all registered users. As shown in Figure 7.2, the groundwork of the system's permission model is laid by three additional primitives. The class *User* represents a user of the system and encapsulates the user's credentials. A *Permission* is a specific permission regarding a certain primitive that can be granted or revoked. The system is able to distinguish between four different kinds of permissions: *Create*, *Read*, *Update* and *Delete*. The semantics of *read*, *update* and *delete* permissions are straight forward. If the according permission is granted to a user, she is able to read, update or delete the information associated with the primitive. In case of sources, this includes the actual data provided by the source as well as its configuration. In case of transformations or extractions, this covers all of the primitives configuration data. The *create* permission on a primitive states that the user is allowed to add child elements to it and is therefore only available for sites (including the global node). A *Role* represents a set of a permissions for a set of primitives. Each user's permissions are modeled as an aggregation of a set of roles. Because rights, roles, groups and users are also primitives from the data architecture, the same mechanism can be used for managing the whole permission model itself.

Because data transformation scripts are able to alter the dataspace, they are also subject to permission management. To this end, scripts conceptually extend the user class, which means that it is possible to associate them with roles. When a script is executed, it accesses the dataspace via the scripting environment. If it tries to create, read or write a dataset for which it does not have the required permissions, an access violation error is raised and the script is terminated. When defining the preamble of a script (i.e., the specification of its input and output behaviour), a data integrator is only able to choose databases in conformance to the script's associated roles. To enable the specification of the databases created by a script, these can be added to the sites in the same way as regular sources but with a flag which indicates that this database is to be generated by a transformation script.

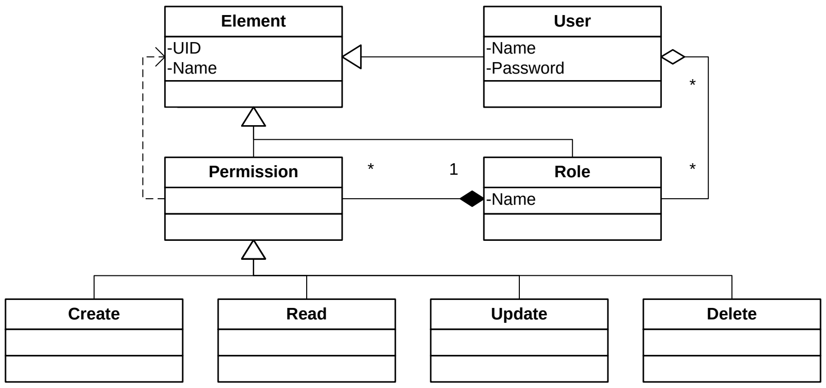


Figure 7.2: UML class diagram of the permission model

Because the permissions of the users can only be altered by data administrators (which have create, update and delete rights on all permission management primitives), there is a central point at which changing access rights can be controlled, monitored and documented. Moreover, the model allows to represent permissions on a very fine-grained level if the data is partitioned accordingly. For example, the data of a research-specific collection could be separated into pseudonymized identifying data and medical data as well as a set of links between them. It would then be possible to define an administrative role for the according site which is allowed to create and delete datasets but is not allowed to view their contents. Additionally, it would be possible to distinguish between users which are allowed read-only access to either the identifying data, the medical data or the links between them. Another role could model users which are allowed to annotate the datasets. By combining these roles, fine-grained permission structures can be modelled that adhere to legal requirements and respect the intellectual property rights of data owners.

7.1.3 System Architecture

The internal architecture of the services deployed on the distributed computing nodes is shown in Figure 7.3. They extend the mediator/wrapper architecture from Figure 5.26 in Chapter 5, which enables distributed query processing. All further components are built around this core functionality. Each site is required to host at least one wrapper instance, which is responsible for performing all local operations such as reading, writing, editing, extracting, transforming, indexing and querying the local datasets. The mediator's task is to manage all data primitives and configuration information, such as data extraction specifications and transformation scripts. The mediator also maintains the global index (see Chapter 5) and provides the global view on all data sources via a querying interface. The mediator and the wrappers communicate via a network protocol. This protocol consists of several classes of messages which are associated to the different functional units. Between each pair of wrapper and mediator a single connection is utilized and messages are classified according to their class and forwarded to the relevant component for further processing. Similar to the distributed querying engine, all of these components are multithreaded and therefore able to process several requests in parallel. Access permissions are solely enforced by the mediator. It requires the users to authenticate themselves and ensures that they are only able to perform operations in conformance with the permission model.

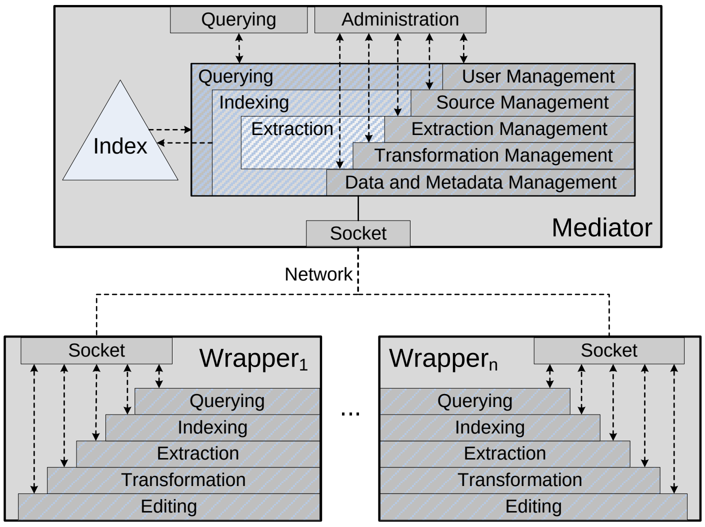


Figure 7.3: Mediator and wrapper architecture

Figure 7.4 shows a layered representation of how these different functional units depend on each other. RDF datasets can either be user-generated, provided by the data extraction layer, or generated by executing data transformation workflows. The data access layer is utilized by all higher-level layers for accessing the datasets. A graphical interface utilizes four functional units, which are integrated with the lower-level layers. An access control component is integrated by all layers.

Wrappers: Regarding data and metadata management, each wrapper provides means to create, read, write and update its local RDF data sources. In addition, each site is able to expose an arbitrary number of non-RDF data sources. These data sources can be exported into an RDF representation by applying the techniques presented in Chapter 4. Any data source which is to be exported has to be accessible from the physical machine, which hosts the wrapper that exposes the site to the mediator. The wrappers are able to execute data extraction processes based on mapping definitions provided by the mediator. To assist the user in building such mapping definitions (e.g., by automatically generating an initial mapping), the wrappers are able to export schema information from local RDBMSs to the mediator. For data transformation purposes each wrapper is also able to execute a provided transformation script. If a wrapper needs to issue a query (i.e., while executing a transformation script), it sends a request to the mediator. The mediator optimizes the query and initiates query processing but informs the participating wrappers that the local results are to be sent to the wrapper that issued the query. Because each wrapper contains a local copy of the global prefix tree and the query execution component, it is able to compute the global result from the local results. As described in Chapter 6 the transformation rules are distributed amongst the wrappers by the mediator in order to perform load balancing. The indexing and querying components provided by the wrappers are similar to the components described in Chapter 5 for distributed query processing. The functionality of the wrappers can be summarized as enabling access to different functional units, which are executed locally. In contrast, the mediator is responsible for utilizing these units for querying, maintaining and semantically integrating the dataspace.

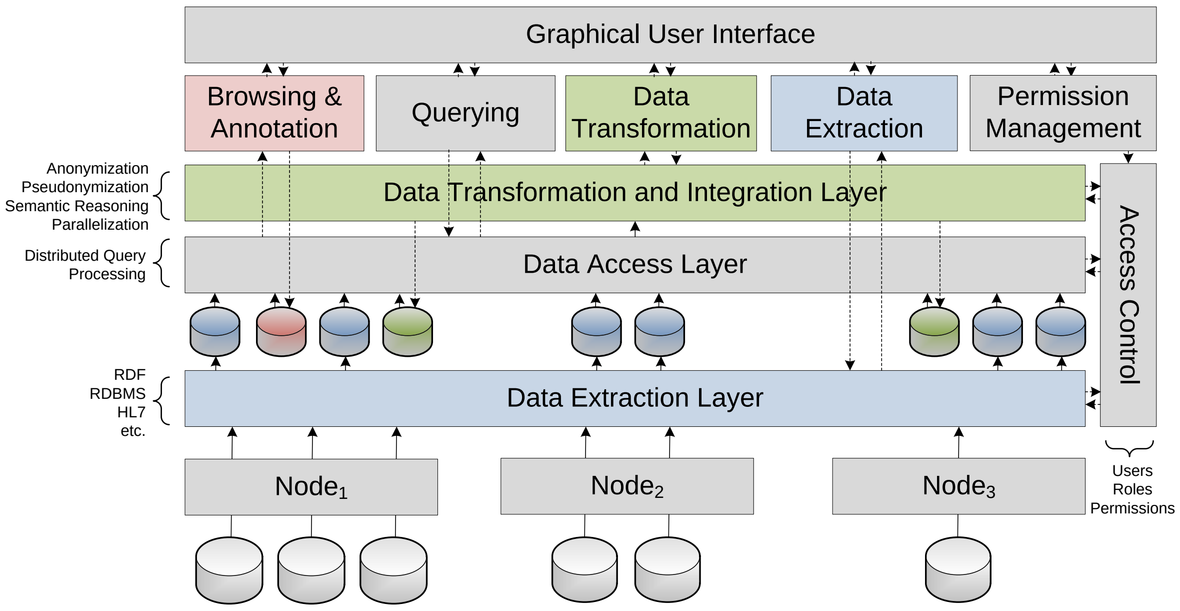


Figure 7.4: Conceptual architecture

Mediator: Firstly, on a meta-level, the mediator enables data integrators to manage the dataspace by configuring the access to all data sources and defining transformation scripts. This includes all metadata, i.e., the definition and specification of sites, data extraction and transformation processes as well as their hierarchical structure. It enforces the permission model presented in the previous section and provides means to create new users and roles as well as to assign permissions for the primitives in the dataspace. Secondly, regarding the execution of data annotation, extraction and transformation processes, the mediator acts as an orchestration component. To this end, it ensures the consistency of the data by considering dependencies between the individual integration steps and guaranteeing atomicity. Thirdly, the mediator is also responsible for providing consistency via concurrency control, e.g., in a multi-user environment. In our prototype we make some simplifying assumptions. To this end, we assume that the dataspace is maintained in two consecutive phases, which are executed repeatedly. These phases also resemble the circular methodology presented in Figure 3.2 in Chapter 3 as the conceptual basis of the incremental ontology-based integration approach:

1. *Export, transform and index (ETI)*: In this phase, all data extraction and transformation processes are executed and the resulting state of the dataspace is indexed.
2. *Browse, query and edit (BQE)*: In this phase, data integrators perform semantic integration tasks by annotating the datasets, providing transformation scripts or adding new datasets.

During the execution of the *ETI* process, the system still provides read access to the previous state of the dataspace. This step can either be executed automatically in predefined intervals (e.g., once every 24 hours) in order to incorporate changes in the underlying databases. Or it can be executed manually if it is required to immediately include new or updated datasets, annotations or transformation scripts. During the *BQE* phase, datasets can either be read (queried) by multiple users, or updated by one user. In this context, write access means that a dataset is opened for browsing

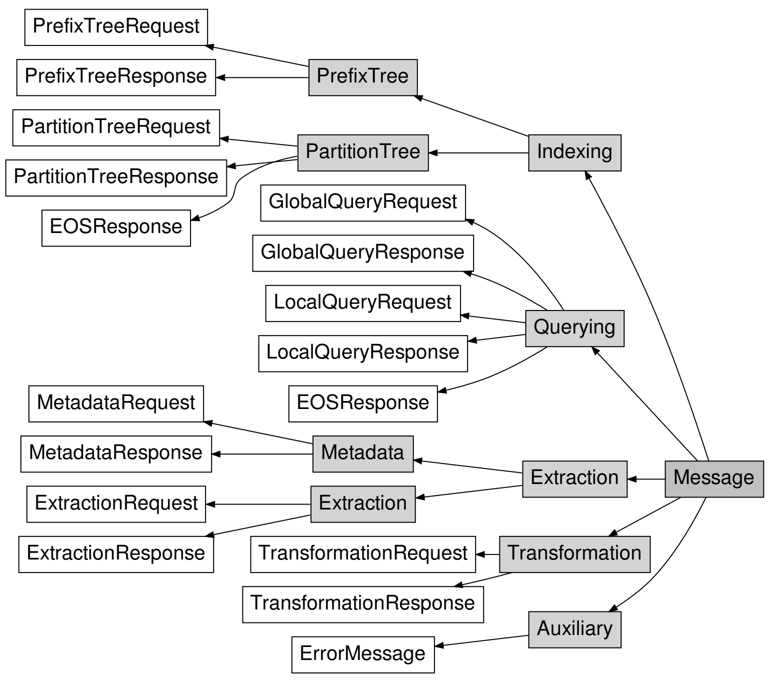


Figure 7.5: Important message types of the network protocol

and editing. It is recommended to store user-provided annotations in additional databases separated from the primary data. In this way, they are not overwritten during subsequent data extraction steps. Additionally, exported data is not altered externally, which resembles a traditional federated solution in which the local databases are accessed directly. Due to this design, it is not possible to concurrently edit a dataset.

Another important functionality of the mediator is the orchestration of data extraction and data transformation processes during the execution of the *ETI* process. To this end, the mediator utilizes the optimizations presented in Chapter 6 to parallelize the execution of transformation scripts and to utilize all computing nodes in the distributed system. The mediator also ensures that the resulting datasets are indexed before they are accessed by any subsequent transformation script. Moreover, the mediator incorporates the data extraction operators into this process in order to ensure that all data required by the transformation scripts is available prior to their execution.

7.1.4 Network Protocol

The mediator and the wrappers communicate via a tailored network protocol which consists of several classes of messages as shown in Figure 7.5. Each class is associated to a certain functional unit. Incoming messages are classified accordingly and forwarded to the relevant component. Regarding the top level of the taxonomy, messages are utilized for *Indexing*, *Querying*, *Extraction*, *Transformation* or *Auxiliary* functionality such as error handling.

For indexing, the system further distinguishes between messages which are utilized to build the global prefix tree (see Chapter 5) and messages which are required to build the partition tree. When building the prefix tree, the system sends a *PrefixTreeRequest* to each wrapper, which is answered by a *PrefixTreeResponse* that contains the local prefix tree. When building the global index, the mediator sends a *PartitionTreeRequest* to each wrapper. This request includes the previously computed global prefix tree, which is stored by the wrapper for reducing the volume of intermediate results during query processing. Each wrapper then responds with a series of *PartitionTreeResponse* messages, each of which contains the partition tree for one partition in the local index. The termination of the local index generation process is signaled by sending an *EOSResponse* (End Of Stream).

Query processing relies on four different classes of messages. A *LocalQueryRequest* message is sent by the mediator in order to request the local execution of a given SPARQL query. The request also includes the reducer generated by the mediator during query optimization (see Chapter 5). The wrapper implements row blocking by responding with a set of *LocalQueryResponse* messages, each of which contains a set of tuples with a predefined maximum size. When all tuples have been sent, the wrapper sends an *EOSResponse*. When a remote site needs to issue a global query while executing a transformation script, it sends a *GlobalQueryRequest* message to the mediator. This message contains the query, an optional set of relevant datasets and the script's unique identifier. The identifier is required to check whether the script is allowed to access the requested datasets. In this case, the mediator optimizes the query and initiates query execution by sending *LocalQueryRequest* messages to the according wrappers. These messages further contain the address of the wrapper which originally issued the query and is responsible for computing the global result. To the site, the mediator will answer with a *GlobalQueryResponse* message which contains the query execution plan to allow the site to orchestrate the execution of bind joins. The wrapper will receive all subsequent *LocalQueryResponse* and *EOSResponse* messages.

For defining and executing data extraction processes the system implements four different types of messages. During the creation of a mapping definition, the mediator can request a description of the schema of a local RDBMS by sending a *MetadataRequest* message. This message needs to define and endpoint at which the local database is reachable via JDBC by the wrapper. The wrapper replies with a *MetadataResponse* which encapsulates the according relational schema. The same messages can also be utilized to ensure that the specification of a native local RDF database (i.e., its path or SPARQL endpoint) is correct, or to ensure that a folder which is to be monitored for incoming Hl7 messages does exist. For the execution of a data extraction process, the mediator sends an *ExtractionRequest* which contains the complete mapping definition. When data extraction has terminated the wrapper replies with an *ExtractionResponse*.

Two classes of messages are utilized to orchestrate the execution of data transformation scripts. The execution of a single script can be started at a wrapper by sending a *TransformationRequest* message. When the script has been executed, the wrapper replies with a *TransformationResponse*. An *ErrorResponse* can be sent by any component to signal a problem during the handling of a request from the mediator.

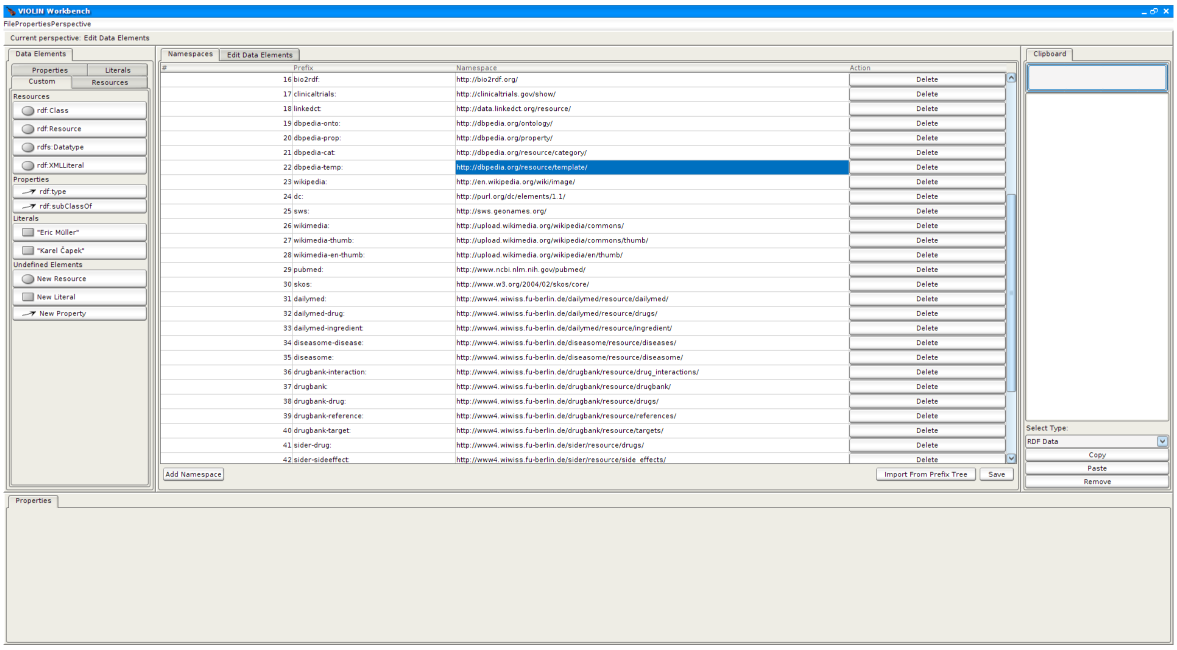
7.2 A Graphical User Interface [1]

In this section we present the VIOLIN Workbench. It implements a graphical user interface to the components provided by the mediator and allows to manage all aspects of the dataspace. It is organized into different perspectives, each of which allows to access one of the functional components. The workbench is organized in panels, which are located at the left, right, bottom or center part of the workbench. The user

[1] Parts of the work presented in this section are based on the student projects [Kuc11], [Kuh11] and [Web12]

interface is separated into a set of modules, which can be attached to the panels. Each perspective consists of a distinct set of modules, which display all relevant information and allow to perform associated actions. The modules themselves are connected via a publisher-subscriber mechanism. Each component registers for a certain set of events and can thus react to changes triggered by the other modules. The workbench requires user authentication and displays information in conformance to the granted permissions. Users which are a member of the group *Administrators* are allowed to access and alter all data and meta-data managed by the integration system. This means that, e.g., a user is only able to see a site, dataset, transformation or extraction for which at least read access has been granted. Each perspective shows an overview over all relevant primitives which are visible to the user. This includes all sites, extractions, transformations, datasets and metadata. These are displayed in a tree-structured hierarchical view. The workbench is implemented in the Java programming language and utilizes the Swing component library for its graphical user interface.

Users and Permissions: This perspective is only available to users with administrative rights. It allows the creation and editing of roles, which are a set of permissions regarding a specific set of primitives. To this end, each role is associated to a matrix in which each row is a primitive and each column represents one distinct type of permission (i.e., create, read, update or delete). The perspective is integrated with the hierarchical overview over all primitives. This means that clicking on a primitive in the hierarchical view highlights the according row in the matrix and vice versa. In addition, the perspective allows to create and alter user accounts and to assign roles.

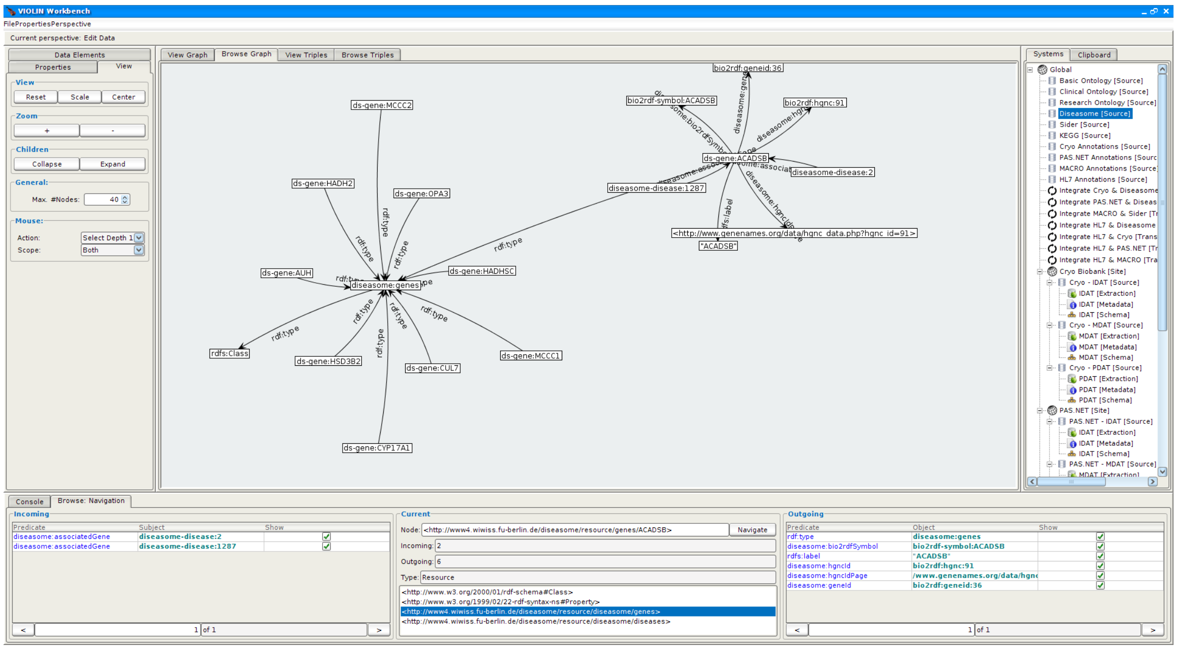


#	Prefix	Namespace	Action
16	bio2rdf:	http://bio2rdf.org/	Delete
17	clinicaltrials:	http://clinicaltrials.gov/show/	Delete
18	linkedct:	http://data.linkedct.org/resource/	Delete
19	dbpedia-onto:	http://dbpedia.org/ontology/	Delete
20	dbpedia-prop:	http://dbpedia.org/property/	Delete
21	dbpedia-cat:	http://dbpedia.org/resource/category/	Delete
22	dbpedia-temp:	http://dbpedia.org/resource/template/	Delete
23	wikipedia:	http://en.wikipedia.org/wiki/image/	Delete
24	dc:	http://purl.org/dc/elements/1.1/	Delete
25	sws:	http://sws.geonames.org/	Delete
26	wikimedia:	http://upload.wikimedia.org/wikipedia/commons/	Delete
27	wikimedia-thumb:	http://upload.wikimedia.org/wikipedia/commons/thumb/	Delete
28	wikimedia-en-thumb:	http://upload.wikimedia.org/wikipedia/en/thumb/	Delete
29	pubmed:	http://www.ncbi.nlm.nih.gov/pubmed/	Delete
30	skos:	http://www.w3.org/2004/02/skos/core/	Delete
31	dailymed:	http://www4.wiwiss.fu-berlin.de/dailymed/resource/dailymed/	Delete
32	dailymed-drug:	http://www4.wiwiss.fu-berlin.de/dailymed/resource/drugs/	Delete
33	dailymed-ingredient:	http://www4.wiwiss.fu-berlin.de/dailymed/resource/ingredient/	Delete
34	diseasome-disease:	http://www4.wiwiss.fu-berlin.de/diseasome/resource/diseases/	Delete
35	diseasome:	http://www4.wiwiss.fu-berlin.de/diseasome/resource/diseasome/	Delete
36	drugbank-interaction:	http://www4.wiwiss.fu-berlin.de/drugbank/resource/drug_interactions/	Delete
37	drugbank:	http://www4.wiwiss.fu-berlin.de/drugbank/resource/drugbank/	Delete
38	drugbank-drug:	http://www4.wiwiss.fu-berlin.de/drugbank/resource/drugs/	Delete
39	drugbank-reference:	http://www4.wiwiss.fu-berlin.de/drugbank/resource/references/	Delete
40	drugbank-target:	http://www4.wiwiss.fu-berlin.de/drugbank/resource/targets/	Delete
41	sider-drug:	http://www4.wiwiss.fu-berlin.de/sider/resource/drugs/	Delete
42	sider-sideeffect:	http://www4.wiwiss.fu-berlin.de/sider/resource/side_effects/	Delete

Figure 7.6: Editing data elements

Data Elements: A screenshot of this perspective is shown in Figure 7.6. It allows to manage a set of data items and prefixes, which are utilized in all other perspectives. This includes the definition of common resources, properties or literals which are then available for editing and creating datasets and annotations. Additionally, it allows to define prefixes, which are utilized to more compactly represent URIs when browsing, editing or querying datasets. It is also fully integrated with the global index structure.

This means that the perspective automatically recommends abbreviations for the most common prefixes in the dataspace, which are determined during the indexing process. For other perspectives, a comprehensive toolbox is provided, which contains all predefined namespaces, resources and literals, e.g., for inserting them into a dataset. The defined data elements are shared by all users of the system.



Incoming

Predicate	Subject	Show
diseasome:associatedGene	diseasome-disease:2	✓
diseasome:associatedGene	diseasome-disease:1287	✓

1 of 1

Current

Node: <http://www4.wiwiss.fu-berlin.de/diseasome/resource/genes/ACADSB>
Incoming: 2
Outgoing: 6
Type: Resource

<http://www.w3.org/2000/01/rdf-schema#Class>
<http://www.w3.org/1999/02/22-rdf-syntax-ns#Property>
<http://www4.wiwiss.fu-berlin.de/diseasome/resource/diseasome/genes>
<http://www4.wiwiss.fu-berlin.de/diseasome/resource/diseasome/diseases>

Outgoing

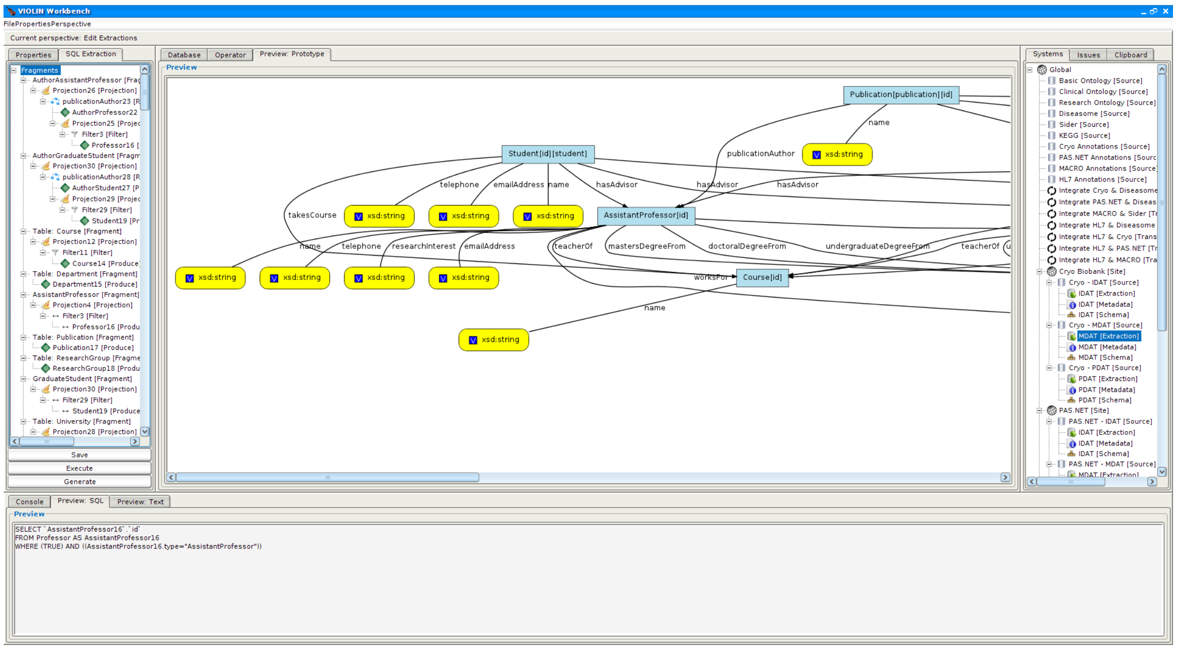
Predicate	Object	Show
rdf:type	diseasome:genes	✓
diseasome:bio2rdfSymbol	bio2rdf-symbol:ACADSB	✓
rdfs:label	"ACADSB"	✓
diseasome:hgncId	bio2rdf:hgnc:91	✓
diseasome:hgncIdPage	/www.genenames.org/data/hgnc	✓
diseasome:geneId	bio2rdf:geneid:36	✓

1 of 1

Figure 7.7: Browsing and editing the dataspace

Data and Metadata: This perspective gives access to all components which allow data integrators to query, browse, edit and annotate data and metadata in the dataspace. It contains a query interface and an editor for low-volume RDF graphs. It also provides a browsing component, which allows to browse and edit large distributed RDF datasets in an integrated manner. Depending on the size of a dataset, the system automatically decides whether it can be opened in the integrated editor or is available for browsing only. The browser not only allows to view a single dataset but also offers an integrated view on an arbitrary subset of the data in the dataspace. Browsing is supported by a module which displays and enables editing of the incoming and outgoing edges of the currently selected node. The toolbox containing the available data elements is integrated for editing and inserting data. A screenshot of this component is shown in Figure 7.7. For data integrators with the according permissions, this perspective also provides means to edit the hierarchical view over all primitives, including means to add, edit or remove available sites and datasets. The graph editor and the browsing component are implemented based on the Java Universal Network/Graph Framework [MFS+05].

Data Extractions: Within the data extraction perspective it is possible to edit the data extraction processes for relational databases and HL7 message streams as well as to add existing RDF datasets to the dataspace. Figure 7.8 shows a screenshot of the most complex of these operations, i.e., editing a mapping definition for a relational database. Here, the left panel displays the mapping definition and allows to add, edit and remove fragments as well as operators. These fragments can be edited in the center panel, which also allows to view the resulting schema for the currently selected operator (as shown in the screenshot). The RDF schema definition is rendered with Graphviz [EGK+02]. In the bottom panel, the SQL expression, which results from compiling the subgraph defined by the currently selected operator, is displayed. Further modules for extracting data from SQL databases allow to edit the vocabulary of the resulting RDF graph by altering the generated production rules and to define the endpoint for the JDBC connection.

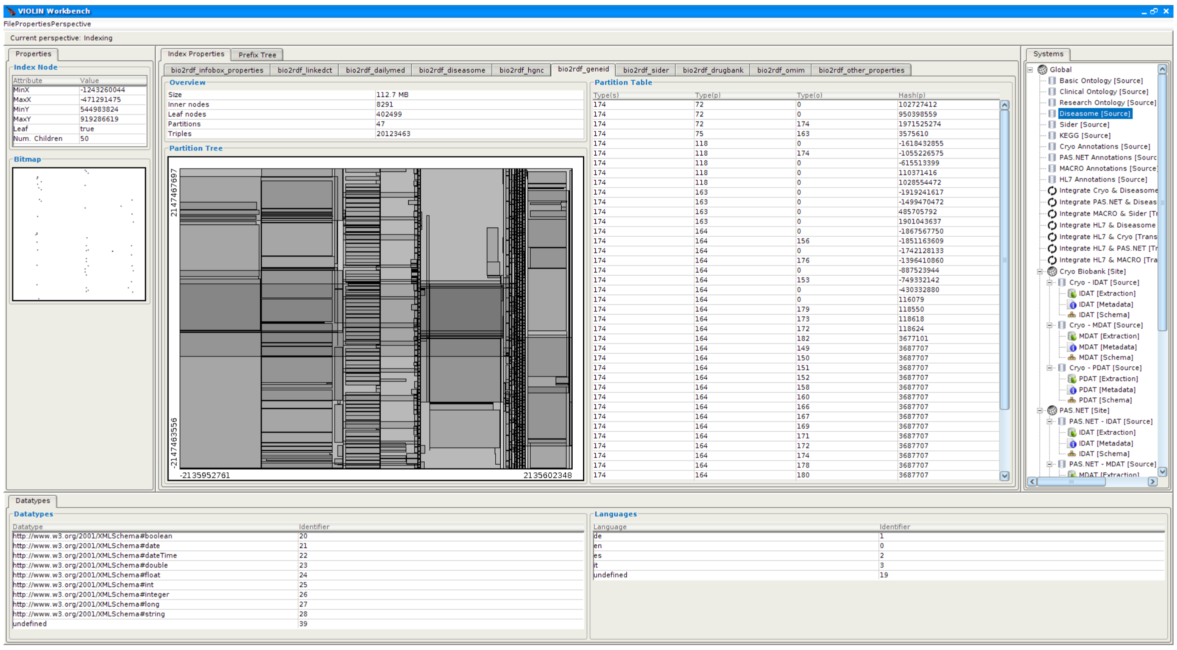


```
SELECT `AssistantProfessor16`.`id`
FROM Professor AS AssistantProfessor16
WHERE (TRUE) AND ((AssistantProfessor16.type="AssistantProfessor"))
```

Figure 7.8: Extracting data from a SQL database

Data Transformations: This perspective allows to edit and create data transformation scripts. To this end, it provides a module containing a text editor for Javascript code and a graphical editor for the definition of the scripts' preambles. Another module allows to manually alter the dependency graph, which is utilized for generating a parallelized global execution plan. Finally, a module allows to view the resulting execution plan and highlights all transformation scripts which were terminated because of local errors during the last ETI phase. Editing the dependency graph is implemented with the Java Universal Network/Graph Framework [MFS+05] and the execution plan is rendered with Graphviz [EGK+02].

System Internals: Finally, this perspective displays system internals for demonstration and debugging purposes. This includes an overview over the system status, consisting of the volume of available main memory, the number of active threads, network statistics and the current state of each node. Moreover, it is possible to visually browse the global synopsis, including the global prefix tree, the partitions, partition trees and bitmasks for each dataset. An example is shown in Figure 7.9. Here, the left panel displays information about the currently selected leaf node in the partition tree, which is shown in the center panel. This panel allows to switch between the datasets via tabs and shows a list of partitions for each dataset. The bottom panel displays the identifiers of all known literal data types.



Index Node

Attribute	Value
MinX	-1243260044
MaxX	-471291475
MinY	544983824
MaxY	919286619
Leaf	true
Num. Children	50

Overview

Size	112.7 MB
Inner nodes	8291
Leaf nodes	402499
Partitions	47
Triples	20123463

Partition Table

Type(s)	Type(p)	Type(o)	Hash(p)
174	72	0	102727412
174	72	0	950398559
174	72	174	1971525274
174	75	163	3575610
174	118	0	-1618432855
174	118	174	-1055226575
174	118	0	-615513399
174	118	0	110371416
174	118	0	1028554472
174	163	0	-1919241617
174	163	0	-1499470472
174	163	0	485705792
174	163	0	1901043637
174	164	0	-1867567750
174	164	156	-1851163609
174	164	0	-1742128133
174	164	176	-1396410860
174	164	0	-887523944
174	164	153	-749332142
174	164	0	-430332880
174	164	0	116079
174	164	179	118550
174	164	173	118618
174	164	172	118624
174	164	182	3677101
174	164	149	3687707
174	164	150	3687707
174	164	151	3687707
174	164	152	3687707
174	164	158	3687707
174	164	160	3687707
174	164	166	3687707
174	164	167	3687707
174	164	169	3687707
174	164	171	3687707
174	164	172	3687707
174	164	174	3687707
174	164	178	3687707
174	164	180	3687707

```
Global
    Basic Ontology [Source]
    Clinical Ontology [Source]
    Research Ontology [Source]
    Diseasome [Source]
    Sider [Source]
    KEGG [Source]
    Cryo Annotations [Source]
    PAS.NET Annotations [Sourc
    MACRO Annotations [Source]
    HL7 Annotations [Source]
    Integrate Cryo & Diseasome
    Integrate PAS.NET & Diseas
    Integrate MACRO & Sider [Tr
    Integrate HL7 & Diseasome
    Integrate HL7 & Cryo [Trans
    Integrate HL7 & PAS.NET [Tr
    Integrate HL7 & MACRO [Tra
    Cryo Biobank [Site]
        Cryo - IDAT [Source]
            IDAT [Extraction]
            IDAT [Metadata]
            IDAT [Schema]
        Cryo - MDAT [Source]
            MDAT [Extraction]
            MDAT [Metadata]
            MDAT [Schema]
        Cryo - PDAT [Source]
            PDAT [Extraction]
            PDAT [Metadata]
            PDAT [Schema]
    PAS.NET [Site]
        PAS.NET - IDAT [Source]
            IDAT [Extraction]
            IDAT [Metadata]
            IDAT [Schema]
        PAS.NET - MDAT [Source]
            MDAT [Extraction]
```

Datatypes

Datatype	Identifier
http://www.w3.org/2001/XMLSchema#boolean	20
http://www.w3.org/2001/XMLSchema#date	21
http://www.w3.org/2001/XMLSchema#dateTime	22
http://www.w3.org/2001/XMLSchema#double	23
http://www.w3.org/2001/XMLSchema#float	24
http://www.w3.org/2001/XMLSchema#int	25
http://www.w3.org/2001/XMLSchema#integer	26
http://www.w3.org/2001/XMLSchema#long	27
http://www.w3.org/2001/XMLSchema#string	28
undefined	39

Languages

Language	Identifier
de	1
en	0
es	2
it	3
undefined	19

Figure 7.9: Browsing the index structure

7.3 Summary and Perspectives

In this chapter, a prototypical system has been proposed which unifies the individual building blocks into a common information integration architecture. The described functionalities and the graphical user interface are oriented towards data integrators and provide means to manage and administrate all aspects of the dataspace. The only interface that is provided for end-users or applications is a SPARQL endpoint that allows to query the dataspace.

In future work it would be important to investigate which further interfaces could be useful for end-users. This includes, e.g., the ability of the user to give feedback about the data provided and to provide annotations and metadata. To this end, successful frontends for researchers, such as the i2b2 workbench, could serve as a starting point. In the spirit of dataspace applications, this interface could also aim at leveraging user interactions for semantic integration. Firstly, this can be implemented implicitly, e.g., by evaluating the acceptance of results sets returned by the querying enginge. Secondly, the users could be asked to give explicit feedback, e.g, by actively ranking results according to their preferences.

The prototypical implementation is a standalone application, that communicates with the integration system via a network interface but also implements some application logic, e.g., caching intermediate results when browsing the dataspace. This design decision is mainly motivated by the comprehensive data visualization features that can be provided in a rich client setup. For real-world applications, especially in a distributed environment, a web-based implementation that consequently implements the Model View Controller (MVC) design pattern could be beneficial in many ways. Firstly, transactional guarantees can be provided more easily if all application logic is clearly separated from the rest of the system. Secondly, the system becomes independent of client-side hardware and can be scaled up on the server-side. Finally, software updates can be distributed in a more reasonable manner which ensures that all users are running the latest version of the frontend.

CHAPTER 8

Summary and Outlook

In this thesis, a first step towards a comprehensive and incremental ontology-based integration solution for translational medical research has been proposed. The concept has been implemented based upon generic and domain-specific components as well as the RDF data model. The major contributions include novel techniques for indexing and querying distributed autonomous RDF databases, for transforming relational databases into an RDF representation, for executing data transformation scripts in a distributed environment and for de-identifying biomedical datasets. We have further presented an overall architecture which integrates the individual components and proposed a comprehensive graphical user interface for data integrators.

The prototype has been tailored to meet the specific requirements of the application domain (see Section 2.2) and the most common use cases (see Section 2.1). It supports data retrieval and the integration of knowledge bases. To this end, the system allows to access important primary data from research and patient care. This data can be annotated and integrated with existing domain-specific metadata, as the expressive RDF data model is frequently used for knowledge bases or ontologies. Furthermore, the graph-structured data model is a natural representation for several types of biomolecular data, such as metabolic pathways. Patient recruitment can be supported by estimating patient accrual rates based on comprehensive information from patient care and clinical research. The system is able to access clinical information systems, such as ADT Systems, Clinical Repositories, Pathology Information Systems or Laboratory Information Systems, which are implemented on top of relational databases. It is further able to retrieve data via the HL7 messaging standard, which is frequently used in clinical environments. Clinical systems provide important data for the characterization of patient cohorts, such as demographics, diagnoses, procedures or laboratory results. Research systems can be utilized to determine the availability of biosamples, whether a patient is already enrolled in a trial (Clinical Trial Management System, CTMS) or to include highly-structured medical data (Clinical Data Management System, CDMS). Even if a system does not encode information in a natural representation (e.g., utilizing the EAV model), the tools presented in this thesis can be utilized to transform the data accordingly.

Moreover, the system offers flexible interfaces for the incremental manual annotation of data. Powerful data de-identification and pseudonymization algorithms are included and allow data integrators to ensure that the patients' privacy is protected. In addition, the separation of data can be utilized to implement fine-grained permission models, which balance the researchers' needs to share data and their willingness to maintain full control over their local datasets.

The prototypical implementation presented in this thesis has been developed for presentation purposes only and is not intended to be deployed for real-world use cases. Several directions for further research regarding the individual components have been described in Sections 4.6, 5.6 and 6.5. For the overall concept and implementation approach, further challenges especially arise in the context of data security and privacy. The prototype assumes that access rights are complex but remain rather static. In some scenarios, by contrast, it might be necessary to incorporate highly dynamic authorization structures. In this case, it would be required to access the information systems on application level and to retrieve data via dedicated interfaces [WLP+09]. A related concept for integrating Webservices into an RDF graph has been presented in [PKS+10].

Due to the incremental methodology the dataspace is never completely integrated or its state is not entirely known. Queries might therefore return inconsistent results. However, this is still useful for many application scenarios. For example, an incomplete result might still corroborate a hypothesis when estimating accrual rates for clinical trials. It can be assumed that a high degree of consistency and semantic integration has to be provided for most use cases, which requires in-depth domain-knowledge from data integrators. To lower these efforts, future work could investigate techniques for learning relationships about the data from user feedback [FHM05, BPE+10, BPF+11].

This thesis focussed on conceptual and technical aspects of an integration solution for translational medical research. For future work it is important to evaluate the concept and potential implementations within the scope of a concrete application scenario and concrete use cases. This includes aspects of user acceptance and the usability of the interface for data integrators. Besides impulses given for related research areas, the concepts and techniques proposed in this thesis can act as a stimulus for the further development of novel types of integration solutions that are thoroughly oriented towards the requirements of the application domain. Although it is unlikely that the diverse types of use cases allow for implementing a single solution that fits all requirements, the domain could benefit from more comprehensive approaches. This could ultimately help to accelerate the translational cycle, from bench to bedside to community and back.

Bibliography

[AAT08] Joan S. Ash, Nicholas R. Anderson, and Peter Tarczy-Hornoch. People and organizational issues in research systems implementation. *Journal of the American Medical Informatics Association*, 15(3):283–289, May 2008.

[ABB$^+$04] A. Arasu, B. Babcock, S. Babu, J. Cieslewicz, M. Datar, K. Ito, R Motwani, U. Srivastava, and J. Widom. STREAM: The stanford data stream management system. Technical Report 2004-20, Stanford University, 2004.

[ABB$^+$07] Ashiq Anjum, Peter Bloodsworth, Andrew Branson, Tamas Hauer, Richard McClatchey, Kamran Munir, Dimtry Rogulin, and Jetendr Shamdasani. The requirements for ontologies in medical data integration: A case study. In *Proceedings of the 11th International Database Engineering and Applications Symposium*, 2007.

[ADB04] Jos Aarts, Hans Doorewaard, and Marc Berg. Understanding implementation: the case of a computerized physician order entry system in a large dutch university medical center. *Journal of the American Medical Informatics Association*, 11(3):207–216, June 2004.

[ADL$^+$09] Sören Auer, Sebastian Dietzold, Jens Lehmann, Sebastian Hellmann, and David Aumüller. Triplify: light-weight linked data publication from relational databases. *Proceedings of the 18th International World Wide Web Conference*, 2009.

[AFK$^+$05] Gagan Aggarwal, Tomas Feder, Krishnaram Kenthapadi, Rajeev Motwani, Rina Panigrahy, Dilys Thomas, and An Zhu. Approximation algorithms for k-Anonymity. In *Proceedings of the 10th International Conference on Database Theory*, 2005.

[AFMPdlF11] Mario Arias, Javier D. Fernández, Miguel A. Martínez-Prieto, and Pablo de la Fuente. An empirical study of real-world SPARQL queries. *CoRR*, abs/1103.5043, 2011.

[AMMH07] Daniel J. Abadi, Adam Marcus, Samuel Madden, and Katherine J. Hollenbach. Scalable semantic web data management using vertical partitioning. In *Proceedings of the 33rd International Conference on Very Large Data Bases*, pages 411–422, 2007.

[APF$^+$10] Gagan Aggarwal, Rina Panigrahy, Tomás Feder, Dilys Thomas, Krishnaram Kenthapadi, Samir Khuller, and An Zhu. Achieving anonymity via clustering. *ACM Transactions on Algorithms*, 6(3), 2010.

[BA05] Roberto J. Bayardo and Rakesh Agrawal. Data privacy through optimal k-Anonymization. In *Proceedings of the 21st International Conference on Data Engineering*, 2005.

[Bar07] Albert-László Barabási. Network medicine — from obesity to the "diseasome". *New England Journal of Medicine*, 357(4):404–407, 2007.

[BB10] Cosmin Basca and Abraham Bernstein. Avalanche: putting the spirit of the web back into semantic web querying. In *Proceedings of the 9th International Semantic Web Conference*, 2010.

[BCGP04] Jesus Barrasa, Oscar Corcho, and Asuncion Gomez-Perez. R2O, an extensible and semantically based database-to-ontology mapping language. In *Proceedings of the 2nd Workshop on Semantic Web and Databases*, 2004.

[BEF10] J. Bisbal, G. Engelbrecht, and A. Frangi. Archetype-based semantic mediation: Incremental provisioning of data services. In *Proceedings of the 2010 IEEE 23rd International Symposium on Computer-Based Medical Systems*, 2010.

[BHJV08] Jürgen Bock, Peter Haase, Qiu Ji, and Raphael Volz. Benchmarking OWL reasoners. In *Proceedings of the 2008 Workshop on Advancing Reasoning on the Web*, 2008.

[Biz03] Christian Bizer. D2R map - a database to RDF mapping language. In *Proceedings of the 12th International World Wide Web Conference*, 2003.

[Biz09] Christian Bizer. The emerging web of linked data. *IEEE Intelligent Systems*, 24(5):87–92, October 2009.

[BLHL01] Tim Berners-Lee, James Hendler, and Ora Lassila. The semantic web. *Scientific American*, pages 28–37, may 2001.

[BLNT07] J. Bauckmann, U. Leser, F. Naumann, and V. Tietz. Efficiently detecting inclusion dependencies. In *Proceedings of the 23rd International Conference on Data Engineering*, 2007.

[BML$^+$09] Richard L. Bradshaw, Susan Matney, Oren E. Livne, Bruce E. Bray, Joyce A. Mitchell, and Scott P. Narus. Architecture of a federated query engine for heterogeneous resources. In *Proceedings of the AMIA Annual Symposium*, 2009.

[BNT$^+$08] François Belleau, Marc-Alexandre Nolin, Nicole Tourigny, Philippe Rigault, and Jean Morissette. Bio2RDF: towards a mashup to build bioinformatics knowledge systems. *Journal of Biomedical Informatics*, 41(5):706–716, October 2008.

[BPE$^+$10] Khalid Belhajjame, Norman W. Paton, Suzanne M. Embury, Alvaro A. A. Fernandes, and Cornelia Hedeler. Feedback-based annotation, selection and refinement of schema mappings for dataspaces. In *Proceedings of the 13th International Conference on Extending Database Technology*, 2010.

[BPF$^+$11] Khalid Belhajjame, Norman W. Paton, Alvaro A. A. Fernandes, Cornelia Hedeler, and Suzanne M. Embury. User feedback as a first class citizen in information integration systems. In *Proceedings of the 5th Biennial Conference on Innovative Data Systems Research*, 2011.

[Bre09] Frank Breitling. A standard transformation from XML to RDF via XSLT. *Astronomische Nachrichten*, 7(7):4, 2009.

[BS09] Christian Bizer and Andreas Schultz. The Berlin SPARQL benchmark. *International Journal on Semantic Web and Information Systems*, 5(2):1–24, 2009.

[But08] Atul J. Butte. Translational bioinformatics: coming of age. *Journal of the American Medical Informatics Association*, 15(6):709–714, December 2008.

[CCR] Continuity of Care Record (CCR) Standard. Website. Available online at `http://www.ccrstandard.com/`; visited on July 1st 2012.

[CCSS07] R. Chirkova, D. Chen, F. Sadri, and T. J. Salo. Pay-as-you-go information integration: The semantic model approach. Technical report, NC State University, 2007.

[CDI] CDISC. Website. Available online at `http://www.cdisc.org/`; visited on July 1st 2012.

[CdVFS08] Valentina Ciriani, Sabrina De Capitani di Vimercati, Sara Foresti, and Pierangela Samarati. k-Anonymous data mining: A survey. In *Privacy-Preserving Data Mining*, pages 105–136. Springer, 2008.

[CFM$^+$09] Kei-Hoi Cheung, H. Robert Frost, M. Scott Marshall, Eric Prud'hommeaux, Matthias Samwald, Jun Zhao, and Adrian Paschke. A journey to semantic web query federation in the life sciences. *BMC Bioinformatics*, 10(S-10):10, 2009.

[CRS$^+$07] Michael J. Cafarella, Christopher Ré, Dan Suciu, Oren Etzioni, and Michele Banko. Structured querying of web text: A technical challenge. In *Proceedings of the 3rd Biennial Conference on Innovative Data Systems Research*, 2007.

[CSF+07] M. K. Campbell, C. Snowdon, D. Francis, D. Elbourne, A.M. McDonald, R. Knight, V. Entwistle, J. Garcia, I. Roberts, and A. Grant. Recruitment to randomised trials: strategies for trial enrollment and participation study. the STEPS study. *Health Technology Assessment*, 11(48):iii, ix–105, November 2007.

[CSRL01] Thomas H. Cormen, Clifford Stein, Ronald L. Rivest, and Charles E. Leiserson. *Introduction to Algorithms*. McGraw-Hill Higher Education, 2nd edition, 2001.

[CTS] Clinical and translational science awards: Home. Website. Available online at `http://ctsaweb.org/`; visited on July 1st 2012.

[CVF+06] Dario Cerizza, Emanuele Della Valle, Doug Foxvog, Reto Krummenacher, and Martin Murth. Towards european patient summaries based on triple space computing. In *Proceedings of the 2006 European Conference on eHealth*, pages 143–154, 2006.

[DCtTdK11] Kathrin Dentler, Ronald Cornet, Annette ten Teije, and Nicolette de Keizer. Comparison of reasoners for large ontologies in the OWL 2 EL profile. *Semantic Web*, 2(2):71–87, 2011.

[DDH08] Anish DasSarma, Xin Dong, and Alon Y. Halevy. Bootstrapping pay-as-you-go data integration systems. In *Proceedings of the 2008 ACM SIGMOD International Conference on Management of Data*, 2008.

[DE12] Fida K. Dankar and Khaled El Emam. The application of differential privacy to health data. In *Proceedings of the 15th International Conference on Extending Database Technology*, 2012.

[DH07] Xin Dong and Alon Y. Halevy. Indexing dataspaces. In *Proceedings of the 2007 ACM SIGMOD International Conference on Management of Data*, 2007.

[DK08] J. Dokulil and J. Katreniakova. Navigation in RDF data. In *Proceedings of the 12th International Conference on Information Visualisation*, July 2008.

[DLMT+10] Martin Dugas, Matthias Lange, Carsten Müller-Tidow, Paulus Kirchhof, and Hans-Ulrich Prokosch. Routine data from hospital information systems can support patient recruitment for clinical studies. *Clinical Trials*, 7(2):183–189, April 2010.

[DMM09] Vikrant G. Deshmukh, Stéphane M. Meystre, and Joyce A. Mitchell. Evaluating the informatics for integrating biology and the bedside system for clinical research. *BMC Medical Research Methodology*, 9(1):70, 2009.

[Don07] Xin Dong. *Providing best-effort services in dataspace systems*. PhD thesis, University of Washington, Seattle, WA, USA, 2007.

[DPA] Bundesbeauftragter für den Datenschutz und die Informationsfreiheit - Data Protection Acts. Website. Available online at `http://www.bfdi.bund.de/EN/DataProtectionActs/DataProtectionActs_node.html`; visited on July 1st 2012.

[DS06] Jens-Peter Dittrich and Marcos Antonio Vaz Salles. iDM: a unified and versatile data model for personal dataspace management. In *Proceedings of the 32nd International Conference on Very Large Data Bases*, 2006.

[DW99] Ton DeWaal and Leon Willenborg. Information loss through global recoding and local suppression. In *Special issue on SDC 14*, pages 17–20. Netherlands Official Statistics, 1999.

[Dwo06] Cynthia Dwork. Differential privacy. In *Proceedings of the 33rd International Colloqium on Automata, Languages and Programming*, 2006.

[EDI$^+$09] Khaled El Emam, Fida Kamal Dankar, Romeo Issa, Elizabeth Jonker, Daniel Amyot, Elise Cogo, Jean-Pierre Corriveau, Mark Walker, Sadrul Chowdhury, Regis Vaillancourt, Tyson Roffey, and Jim Bottomley. A globally optimal k-Anonymity method for the de-identification of health data. *Journal of the Amererican Medical Informatics Association*, 16(5):670–682, 2009.

[EGK$^+$02] John Ellson, Emden Gansner, Lefteris Koutsofios, Stephen North, and Gordon Woodhull. Graphviz - open source graph drawing tools. In *Graph Drawing*, volume 2265 of *Lecture Notes in Computer Science*, pages 594–597. Springer Berlin / Heidelberg, 2002.

[EJC$^+$05] Peter J. Embi, Anil Jain, Jeffrey Clark, Susan Bizjack, Richard Hornung, and Martin C. Harris. Effect of a clinical trial alert system on physician participation in trial recruitment. *Archives of Internal Medicine*, 165(19):2272–2277, October 2005.

[EKP09] Peter J. Embi, Stanley E. Kaufman, and Philip R. O. Payne. Biomedical informatics and outcomes research: Enabling knowledge-driven healthcare. *Circulation*, 120(23), December 2009.

[Ema10] Khaled El Emam. Risk-based de-identification of health data. *IEEE Security & Privacy*, 8(3):64–67, 2010.

[EP09] Peter J. Embi and Philip R. O. Payne. Clinical research informatics: challenges, opportunities and definition for an emerging domain. *Journal of the American Medical Informatics Association*, 16(3):316–327, June 2009.

[EPS08] Jérôme Euzenat, Axel Polleres, and François Scharffe. Processing ontology alignments with SPARQL. In *Proceedings of the 2008 International Conference on Complex, Intelligent and Software Intensive Systems*, 2008.

[FBR+10] Isabel Fortier, Paul R Burton, Paula J Robson, Vincent Ferretti, Julian Little, Francois L'Heureux, Mylène Deschênes, Bartha M Knoppers, Dany Doiron, Joost C Keers, Pamela Linksted, Jennifer R Harris, Geneviève Lachance, Catherine Boileau, Nancy L Pedersen, Carol M Hamilton, Kristian Hveem, Marilyn J Borugian, Richard P Gallagher, John McLaughlin, Louise Parker, John D Potter, John Gallacher, Rudolf Kaaks, Bette Liu, Tim Sprosen, Anne Vilain, Susan A Atkinson, Andrea Rengifo, Robin Morton, Andres Metspalu, H Erich Wichmann, Mark Tremblay, Rex L Chisholm, Andrés Garcia-Montero, Hans Hillege, Jan-Eric Litton, Lyle J Palmer, Markus Perola, Bruce H R Wolffenbuttel, Leena Peltonen, and Thomas J Hudson. Quality, quantity and harmony: the DataSHaPER approach to integrating data across bioclinical studies. *International Journal of Epidemiology*, 39(5):1383–1393, October 2010.

[FEHM08] Douglas B. Fridsma, Julie Evans, Smita Hastak, and Charles N. Mead. The BRIDG project: a technical report. *Journal of the American Medical Informatics Association*, 15(2):130–137, April 2008.

[Fei01] Manning Feinleib. A dictionary of epidemiology. *American Journal of Epidemiology*, 154(1):93 –94, July 2001.

[FHM05] Michael Franklin, Alony Y. Halevy, and David Maier. From databases to dataspaces: a new abstraction for information management. *SIGMOD Record*, 34(4):27–33, December 2005.

[FM11] Pit Fender and Guido Moerkotte. A new, highly efficient, and easy to implement top-down join enumeration algorithm. In *Proceedings of the 27th International Conference on Data Engineering*, 2011.

[Fre12] Andre Freitag. Effiziente Parallelisierung von Transformationsworkflows in einer ontologiebasierten Integrationslösung für biomedizinische Daten. Interdisciplinary project (IDP), Technische Universität München, 2012. Supervised by Fabian Praßer.

[FW97] Charles P. Friedman and Jeremy Wyatt. *Evaluation methods in medical informatics*. Springer, 1997.

[FWCY10] Benjamin C. M. Fung, Ke Wang, Rui Chen, and Philip S. Yu. Privacy-preserving data publishing: A survey of recent developments. *ACM Computing Surveys*, 42(4), 2010.

[Gar01] Simson L. Garfinkel. *Database nation - the death of privacy in the 21th century*. O'Reilly, 2001.

[GC07] R. Ghawi and N. Cullot. Database-to-ontology mapping generation for semantic interoperability. In *Proceedings of the 3rd International Workshop on Database Interoperability*, 2007.

[GCP] EMEA ICH Topic E 6 (R1) - guideline for good clinical practice. Website. Available online at http://www.ema.europa.eu/docs/en_GB/document_library/Scientific_guideline/2009/09/WC500002874.pdf; visited on July 1st 2012.

[GCV$^+$07] Kwang-Il Goh, Michael E. Cusick, David Valle, Barton Childs, Marc Vidal, and Albert-László Barabási. The human disease network. *Proceedings of the National Academy of Sciences*, 104(21):8685 –8690, May 2007.

[GHKS08] Bernardo Cuenca Grau, Ian Horrocks, Yevgeny Kazakov, and Ulrike Sattler. Modular reuse of ontologies: Theory and practice. *Journal of Artificial Intelligence Research*, 31:273–318, 2008.

[GMF$^+$03] John H. Gennari, Mark A. Musen, Ray W. Fergerson, William E. Grosso, Monica Crubézy, Henrik Eriksson, Natalya F. Noy, and Samson W. Tu. The evolution of protégé: an environment for knowledge-based systems development. *International Journal of Human-Computer Studies*, 58(1):89–123, January 2003.

[GNTT10] Jeremy Goecks, Anton Nekrutenko, James Taylor, and The Galaxy Team. Galaxy: a comprehensive approach for supporting accessible, reproducible, and transparent computational research in the life sciences. *Genome Biology*, 11(8):R86, August 2010.

[GPH04] Yuanbo Guo, Zhengxiang Pan, and Jeff Heflin. An evaluation of knowledge base systems for large OWL datasets. In *Proceedings of the 3rd International Semantic Web Conference*, 2004.

[GPH05] Yuanbo Guo, Zhengxiang Pan, and Jeff Heflin. LUBM: A benchmark for OWL knowledge base systems. *Journal of Web Semantics*, 3(2-3):158–182, October 2005.

[GRD] GRDDL use cases: Scenarios of extracting RDF data from XML documents. Website. Available online at http://www.w3.org/TR/grddl-scenarios/; visited on July 1st 2012.

[Gru93] Thomas R. Gruber. A translation approach to portable ontology specification. *Knowledge Acquisition*, 5(2):199–220, June 1993.

[GS11] Olaf Görlitz and Steffen Staab. SPLENDID: SPARQL Endpoint Federation Exploiting VOID Descriptions. In *Proceedings of the 2nd International Workshop on Consuming Linked Data*, 2011.

[GTH06] Tom Gardiner, Dmitry Tsarkov, and Ian Horrocks. Framework for an automated comparison of description logic reasoners. In *Proceedings of the 5th International Semantic Web Conference*, 2006.

[Gut84] Antonin Guttman. R-trees: A dynamic index structure for spatial searching. In *Proceedings of the 6th International Conference on Management of Data*, 1984.

[Hal01] Alon Y. Halevy. Answering queries using views: A survey. *VLDB Journal*, 10(4):270–294, 2001.

[HAP] Free and open-source HL7 java parser and library. Website. Available online at `http://hl7api.sourceforge.net/`; visited on July 1st 2012.

[HAR11] Jiewen Huang, Daniel Abadi, and Kun Ren. Scalable SPARQL querying of large RDF graphs. In *Proceedings of the 37th International Conference on Very Large Data Bases*, 2011.

[HB11] Tom Heath and Christian Bizer. *Linked Data: Evolving the Web into a Global Data Space*. Synthesis Lectures on the Semantic Web. Morgan & Claypool Publishers, 2011.

[HBF09] Olaf Hartig, Christian Bizer, and Johann Christoph Freytag. Executing SPARQL queries over the web of linked data. In *Proceedings of the 8th International Semantic Web Conference*, pages 293–309, 2009.

[HCL] Semantic Web Health Care and Life Sciences (HCLS) Interest Group. Website. Available online at `http://www.w3.org/2001/sw/hcls/`; visited on July 1st 2012.

[HER] HermiT reasoner: Home. Website. Available online at `http://hermit-reasoner.com/`; visited on July 1st 2012.

[HFM06] Alon Y. Halevy, Michael J. Franklin, and David Maier. Principles of dataspace systems. In *Proceedings of the 25th ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems*, 2006.

[HHBN10] Shan He, John F. Hurdle, Jeffrey R. Botkin, and Scott P. Narus. Integrating a federated healthcare data query platform with electronic IRB information systems. In *Proceedings of the AMIA Annual Symposium*, volume 2010, pages 291–295, 2010.

[HHK$^+$10] Andreas Harth, Katja Hose, Marcel Karnstedt, Axel Polleres, Kai-Uwe Sattler, and Jürgen Umbrich. Data summaries for on-demand queries over linked data. In *Proceedings of the 19th International World Wide Web Conference*, 2010.

[HHUD07] Aidan Hogan, Andreas Harth, Jürgen Umrich, and Stefan Decker. Towards a scalable search and query engine for the web. In *Proceedings of the 16th International World Wide Web Conference*, 2007.

[HKK$^+$07] Katja Hose, Marcel Karnstedt, Anke Koch, Kai-Uwe Sattler, and Daniel Zinn. Processing rank-aware queries in P2P systems. In *Proceedings of the 2005/2006 International Workshop on Databases, Information Systems, and Peer-to-Peer Computing*, 2007.

[HKL$^+$09] Oktie Hassanzadeh, Anastasios Kementsietsidis, Lipyeow Lim, Renée J. Miller, and Min Wang. Linkedct: A linked data space for clinical trials. *CoRR*, abs/0908.0567, 2009.

[HL7] About health level seven international. Website. Available online at `http://www.hl7.org/about/index.cfm`; visited on July 1st 2012.

[HMRR08] Bill Howe, David Maier, Nicolas Rayner, and James Rucker. Quarrying dataspaces: Schemaless profiling of unfamiliar information sources. In *Proceedings of the 24th International Conference on Data Engineering*, 2008.

[HSR$^+$08] Nils Homer, Szabolcs Szelinger, Margot Redman, David Duggan, Waibhav Tembe, Jill Muehling, John V. Pearson, Dietrich A. Stephan, Stanely F. Nelson, and David W. Craig. Resolving individuals contributing trace amounts of DNA to highly complex mixtures using high-density SNP genotyping microarrays. *PLoS Genetics*, 4(8), August 2008.

[IJB11] Robert Isele, Anja Jentzsch, and Christian Bizer. Efficient multidimensional blocking for link discovery without losing recall. In *Proceedings of the 14th International Workshop on the Web and Databases*, 2011.

[JB04] Dongwon Jeong and Doo-Kwon Baik. Incremental data integration based on hierarchical metadata registry with data visibility. *Information Sciences*, 162(3-4):147–181, June 2004.

[JDG07] Alpa Jain, AnHai Doan, and Luis Gravano. Sql queries over unstructured text databases. In *Proceedings of the 23rd International Conference on Data Engineering*, 2007.

[JEN] Apache jena. Website. Available online at `http://jena.apache.org/`; visited on July 1st 2012.

[JFH08] Shawn R. Jeffery, Michael J. Franklin, and Alon Y. Halevy. Pay-as-you-go user feedback for dataspace systems. In *Proceedings of the 2008 ACM SIGMOD International Conference on Management of Data*, 2008.

[JIB10] Anja Jentzsch, Robert Isele, and Chris Bizer. Silk - generating RDF links while publishing or consuming linked data. In *Proceedings of the 9th International Semantic Web Conference*, November 2010.

[JS12] Priya Jayaratna and Kamran Sartipi. HL7 v3 message extraction using semantic web techniques. *International Journal of Knowledge Engineering and Data Mining*, 2(1):89–115, 2012.

[Kas11] Vipul Kashyap. Semantic web and translational medicine : Creating the next generation healthcare enterprise - asian hospital and healthcare management. Website, 2011. Available online at `http://www.asianhhm.com/information_technology/semantic_web_translational_medicine.htm`; visited on July 1st 2012.

[KB09] Sarah Killcoyne and John Boyle. Managing chaos: lessons learned developing software in the life sciences. *Computing in Science & Engineering*, 11(6):20–29, November 2009.

[KEG] KEGG: Kyoto Encyclopedia of Genes and Genomes. Website. Available online at `http://www.genome.jp/kegg/`; visited on July 1st 2012.

[KG00] M. Kanehisa and S. Goto. KEGG: kyoto encyclopedia of genes and genomes. *Nucleic Acids Research*, 28(1):27–30, January 2000.

[KLK12] Yong-Bin Kang, Yuan-Fang Li, and Shonali Krishnaswamy. A rigorous characterization of reasoning performance – a tale of four reasoners. In *Proceedings of the 2012 OWL Reasoner Evaluation Workshop*, 2012.

[Kos00] Donald Kossmann. The state of the art in distributed query processing. *ACM Computing Surveys*, 32(4):422–469, 2000.

[KPE$^+$12a] Florian Kohlmayer[1], Fabian Prasser[1], Claudia Eckert, Alfons Kemper, and Klaus A. Kuhn. Flash: efficient, stable and optimal k-Anonymity. In *Proceedings of the 4th IEEE International Conference on Information Privacy, Security, Risk and Trust*, 2012. .

[KPE$^+$12b] Florian Kohlmayer[1], Fabian Prasser[1], Claudia Eckert, Alfons Kemper, and Klaus A. Kuhn. Highly efficient optimal k-Anonymity for biomedical datasets. In *Proceedings of the 25th IEEE International Symposium on Computer-Based Medical Systems*, 2012.

[KRA06] Jan Kunze, Thomas Riechert, and Sören Auer. Eine schnittstelle für arztpraxisdaten mittels einer ontologie auf basis von HL7 version 3. In *Tagungsband XML-Tage*, September 2006.

[KSA$^+$08] Stefan Krompass, Andreas Scholz, Martina-Cezara Albutiu, Harumi A. Kuno, Janet L. Wiener, Umeshwar Dayal, and Alfons Kemper. Quality of service-enabled management of database workloads. *IEEE Data Engineering Bulletin*, 31(1):20–27, 2008.

[KSKR05] Richard Kuntschke, Bernhard Stegmaier, Alfons Kemper, and Angelika Reiser. StreamGlobe: Processing and sharing data streams in grid-based P2P infrastructures. In *VLDB*, 2005.

[Kuc11] Jan Kucera. Implementierung eines leichtgewichtigen Werkzeugs für die Bearbeitung verteilter RDF Datenätze. Interdisciplinary project (IDP), Technische Universität München, 2011. Supervised by Fabian Praßer.

[Kuh11] Martin Kuhn. Implementierung einer generischen Naviagationskomponente für verteilte RDF Datenquellen. Interdisciplinary project (IDP), Technische Universität München, 2011. Supervised by Fabian Praßer.

[Laz02] Yuri Lazebnik. Can a biologist fix a radio?–or, what i learned while studying apoptosis. *Cancer Cell*, 2(3):179–182, September 2002.

[1]Both authors contributed equally to this work

[LDR05] Kristen LeFevre, David J. DeWitt, and Raghu Ramakrishnan. Incognito: Efficient full-domain k-Anonymity. In *Proceedings of the 2005 ACM SIGMOD International Conference on Management of Data*, 2005.

[LDW05] Man Li, Xiaoyong Du, and Shan Wang. A semi-automatic ontology acquisition method for the semantic web. In *Advances in Web-Age Information Management*, volume 3739 of *Lecture Notes in Computer Science*, pages 209–220. Springer Berlin / Heidelberg, 2005.

[LEL97] Scott T. Leutenegger, J. M. Edgington, and Mario A. Lopez. STR: A simple and efficient algorithm for R-tree packing. In *Proceedings of the 13th International Conference on Data Engineering*, 1997.

[LH10] Yingjie Li and Jeff Heflin. Using reformulation trees to optimize queries over distributed heterogeneous sources. In *Proceedings of the 9th International Semantic Web Conference*, 2010.

[LIN] Linked data - design issues. Website. Available online at `http://www.w3.org/DesignIssues/LinkedData.html`; visited on July 1st 2012.

[LK04] Richard Lenz and Klaus A. Kuhn. Towards a continuous evolution and adaptation of information systems in healthcare. *International Journal of Medical Informatics*, 73(1):75–89, February 2004.

[LLV07] Ninghui Li, Tiancheng Li, and Suresh Venkatasubramanian. t-Closeness: Privacy beyond k-Anonymity and l-Diversity. In *Proceedings of the 23rd Interational Conference on Data Engineering*, 2007.

[LMM$^+$05] S. Liu, Wei Ma, R. Moore, V. Ganesan, and S. Nelson. RxNorm: prescription for electronic drug information exchange. *IT Professional*, 7(5), October 2005.

[LMMS$^+$07] Brenton Louie, Peter Mork, Fernando Martin-Sanchez, Alon Y. Halevy, and Peter Tarczy-Hornoch. Data integration and genomic medicine. *Journal of Biomedical Informatics*, 40(1):5–16, February 2007.

[LQS11] Ninghui Li, Wahbeh Qardaji, and Dong Su. Provably private data anonymization: Or, k-Anonymity meets differential privacy. Technical Report TR-2010-27, Purdue University, 2011.

[LT10] Günter Ladwig and Thanh Tran. Linked data query processing strategies. In *Proceedings of the 9th International Semantic Web Conference*, 2010.

[LW09a] Andreas Langegger and Wolfram Woss. RDFStats - an extensible RDF statistics generator and library. In *Proceedings of the 2009 20th International Workshop on Database and Expert Systems Application*, 2009.

[LW09b] Andreas Langegger and Wolfram Wöß. XLWrap — querying and integrating arbitrary spreadsheets with SPARQL. In *Proceedings of the 8th International Semantic Web Conference*, 2009.

[LWB08] Andreas Langegger, Wolfram Wöß, and Martin Blöchl. A semantic web middleware for virtual data integration on the web. In *Proceedings of the 5th European Semantic Web Conference*, 2008.

[LYH12] Yingjie Li, Yang Yu, and Jeff Heflin. Evaluating reasoners under realistic semantic web conditions. In *Proceedings of the 2012 OWL Reasoner Evaluation Workshop*, 2012.

[MBG$^+$09] Konstantinos Makris, Nikos Bikakis, Nektarios Gioldasis, Chrisa Tsinaraki, and Stavros Christodoulakis. Towards a mediator based on OWL and SPARQL. In *Proceedings of the 2nd World Summit on the Knowledge Society*, 2009.

[MFS$^+$05] J. Madadhain, D. Fisher, P. Smyth, S. White, and Y. B. Boey. Analysis and visualization of network data using JUNG. *Journal of Statistical Software*, 10:1–35, 2005.

[Mil06] Jerry L. Miller. The EHR solution to clinical trial recruitment in physician groups. *Health Management Technology*, 27(12):22–25, December 2006.

[MJC$^+$07] Jayant Madhavan, Shawn R. Jeffery, Shirley Cohen, Xin (Luna) Dong, David Ko, Cong Yu, and Alon Halevy. Web-scale data integration: You can only afford to pay as you go. In *Proceedings of the 3rd Biennial Conference on Innovative Data Systems Research*, 2007.

[MKGV07] Ashwin Machanavajjhala, Daniel Kifer, Johannes Gehrke, and Muthuramakrishnan Venkitasubramaniam. l-Diversity: Privacy beyond k-Anonymity. *ACM Transactions on Knowledge Discovery from Data*, 1(1), 2007.

[MKM09] Matthias Löbe Magnus, Knuth, and Ronald Mücke. TIM: A semantic web application for the specification of metadata items in clinical research. In Scott M. Marshall, Albert Burger, Paolo Romano, Adrian Paschke, and Andrea Splendiani, editors, *Proceedings of the Semantic Web Applications and Tools for Life Sciences Workshop*, volume 559, 2009.

[MM04] Frank Manola and Erric Miller. RDF primer. Website, 2004. Available online at `http://www.w3.org/TR/rdf-primer/`; visited on July 1st 2012.

[MPB$^+$09] James P. McCusker, Joshua A. Phillips, Alejandra Beltrán, Anthony Finkelstein, and Michael Krauthammer. Semantic web data warehousing for caGrid. *BMC Bioinformatics*, 10(Suppl 10):S2, 2009.

[MPL07] Juha Muilu, Leena Peltonen, and Jan-Eric Litton. The federated database–a basis for biobank-based post-genome studies, integrating phenome and genome data from 600,000 twin pairs in europe. *European Journal of Human Genetics*, 15(7):718–723, July 2007.

[MS05] B. Motik and R. Studer. Kaon2 - a scalable reasoning tool for the semantic web. In *Proceedings of the 2nd European Semantic Web Conference*, 2005.

[Mus08] Adnan Muslimovic. Design and implementation of a dataspace model for e-science applications. Master's thesis, Universität Wien, 2008.

[MW04] Adam Meyerson and Ryan Williams. On the complexity of optimal k-Anonymity. In *Proceedings of the 32rd ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems*, 2004.

[MWM$^+$10] Shawn N. Murphy, Griffin Weber, Michael Mendis, Vivian Gainer, Henry C. Chueh, Susanne Churchill, and Isaac S. Kohane. Serving the enterprise and beyond with informatics for integrating biology and the bedside (i2b2). *Journal of the American Medical Informatics Association*, 17(2):124–130, April 2010.

[MZV$^+$09] Parsa Mirhaji, Min Zhu, Mattew Vagnoni, Elmer V. Bernstam, Jiajie Zhang, and Jack W. Smith. Ontology driven integration platform for clinical and translational research. *BMC Bioinformatics*, 10(S-2), 2009.

[N$^+$07] Mehmet Ercan Nergiz et al. Hiding the presence of individuals from shared databases. In *Proceedings of the 2007 ACM SIGMOD International Conference on Management of Data*, 2007.

[NBHH08] Andrew Newman, Chris Bouton, Jane Hunter, and Jane Hunter. A scale-out RDF molecule store for distributed processing of biomedical data. In *Proceedings of the 2008 Semantic Web for Health Care and Life Sciences Workshop*, 2008.

[ND08] Wang Ning and Xu De. Resource summary for pay-as-you-go dataspace systems. In *Proceedings of the 9th International Conference on Signal Processing*, 2008.

[NIH] Translational research - overview. Website. Available online at `http://commonfund.nih.gov/clinicalresearch/overview-translational.aspx`; visited on July 1st 2012.

[NM11] Thomas Neumann and Guido Moerkotte. Characteristic sets: Accurate cardinality estimation for RDF queries with multiple joins. In *Proceedings of the 27th International Conference on Data Engineering*, 2011.

[NMC$^+$99] P. M. Nadkarni, L. Marenco, R. Chen, E. Skoufos, G. Shepherd, and P. Miller. Organization of heterogeneous scientific data using the EAV/CR representation. *Journal of the American Medical Informatics Association*, 6(6):478–493, December 1999.

[NS08] Arvind Narayanan and Vitaly Shmatikov. Robust de-anonymization of large sparse datasets. In *Proceedings of the 29th IEEE Symposium on Security and Privacy*, 2008.

[NW09] Thomas Neumann and Gerhard Weikum. Scalable join processing on very large RDF graphs. In *Proceedings of the ACM SIGMOD International Conference on Management of Data*, 2009.

[NW10] Thomas Neumann and Gerhard Weikum. The RDF-3X engine for scalable management of RDF data. *VLDB Journal*, 19(1):91–113, 2010.

[ODC$^+$08] Eyal Oren, Renaud Delbru, Michele Catasta, Richard Cyganiak, Holger Stenzhorn, and Giovanni Tummarello. Sindice.com: a document-oriented lookup index for open linked data. *International Journal of Metadata, Semantics and Ontologies*, 3:37–52, November 2008.

[ODD06] Eyal Oren, Renaud Delbru, and Stefan Decker. Extending faceted navigation for RDF data. In *Proceedings of the 5th International Semantic Web Conference*, 2006.

[OK07] Christian Ohmann and Wolfgang Kuchinke. Meeting the challenges of patient recruitment: A role for electronic health records. *International Journal of Pharmaceutical Medicine*, 21(4):263–270, 2007.

[OLHP$^+$10] Kristian Ovaska, Marko Laakso, Saija Haapa-Paananen, Riku Louhimo, Ping Chen, Viljami Aittomäki, Erkka Valo, Javier Núñez-Fontarnau, Ville Rantanen, Sirkku Karinen, Kari Nousiainen, Anna-Maria Lahesmaa-Korpinen, Minna Miettinen, Lilli Saarinen, Pekka Kohonen, Jianmin Wu, Jukka Westermarck, and Sampsa Hautaniemi. Large-scale data integration framework provides a comprehensive view on glioblastoma multiforme. *Genome Medicine*, 2(9):65, September 2010.

[OMBB$^+$12] Lucila Ohno-Machado, Vineet Bafna, Aziz A Boxwala, Brian E Chapman, Wendy W Chapman, Kamalika Chaudhuri, Michele E Day, Claudiu Farcas, Nathaniel D Heintzman, Xiaoqian Jiang, Hyeoneui Kim, Jihoon Kim, Michael E Matheny, Frederic S Resnic, and Staal A Vinterbo. iDASH: integrating data for analysis, anonymization, and sharing. *Journal of the American Medical Informatics Association*, 19(2):196–201, April 2012.

[OMI] OMIM home. Website. Available online at `http://www.ncbi.nlm.nih.gov/omim`; visited on July 1st 2012.

[OWLa] OWL web ontology language overview. Website. Available online at `http://www.w3.org/TR/owl-features/`; visited on July 1st 2012.

[OWLb] OWL 2 Web Ontology Document Overview. Website. Available online at `http://www.w3.org/TR/owl2-overview/`; visited on July 1st 2012.

[OWLc] OWLIM: Ontotext. Website. Available online at `http://www.ontotext.com/owlim/`; visited on July 1st 2012.

[PAG09] Jorge Pérez, Marcelo Arenas, and Claudio Gutierrez. Semantics and complexity of SPARQL. *ACM Transactions on Database Systems*, 34(3), September 2009.

[PBS$^+$10] Philip R. O. Payne, Tara B. Borlawsky, William Stephens, Matthew C. Barrett, Tri Nguyen-Pham, and Andrew W. Greaves. The TRITON project: Design and implementation of an integrative translational research information management platform. *AMIA Annual Symposium Proceedings*, 2010:617–621, 2010.

[PEL] Pellet: OWL 2 reasoner for java. Website. Available online at `http://clarkparsia.com/pellet/`; visited on July 1st 2012.

[PES09] Philip R. O. Payne, Peter J. Embi, and Chandan K. Sen. Translational informatics: enabling high-throughput research paradigms. *Physiological Genomics*, 39(3):131–140, November 2009.

[PKK12] Fabian Prasser, Alfons Kemper, and Klaus A. Kuhn. Efficient distributed query processing for autonomous RDF databases. In *Proceedings of the 15th International Conference on Extending Database Technology*, 2012.

[PKKK12] Fabian Prasser, Florian Kohlmayer, Alfons Kemper, and Klaus A. Kuhn. A generic transformation of HL7 messages into the Resource Description Framework data model. In *GI-Jahrestagung*, LNI, 2012.

[PKS$^+$10] Nicoleta Preda, Gjergji Kasneci, Fabian M. Suchanek, Thomas Neumann, Wenjun Yuan, and Gerhard Weikum. Active knowledge: dynamically enriching RDF knowledge bases by web services. In *Proceedings of the 2010 ACM SIGMOD International Conference on Management of Data*, 2010.

[PRO] Health level seven. Website. Available online at `http://protege.stanford.edu/ontologies/HL7RIM/`; visited on July 1st 2012.

[PS08] Eric Prud'hommeaux and Andy Seaborne. SPARQL query language for RDF. Website, 2008. Available online at `http://www.w3.org/TR/rdf-sparql-query/`; visited on July 1st 2012.

[PV11] Francois Picalausa and Stijn Vansummeren. What are real SPARQL queries like? In *Proceedings of the 3rd International Workshop on Semantic Web Information Management*, 2011.

[PVS10] E. D. Perakslis, J. VanDam, and S. Szalma. How informatics can potentiate precompetitive open-source collaboration to jump-start drug discovery and development. *Clinical pharmacology and therapeutics*, 87(5):614–616, May 2010.

[PWL$^+$11a] Fabian Prasser, Sebastian H. R. Wurst, Gregor Lamla, Alfons Kemper, and Klaus A. Kuhn. Inkrementelle ontologiebasierte informationsintegration für die translationale medizinische forschung. In *GI-Jahrestagung*, LNI, 2011.

[PWL$^+$11b] Fabian Prasser, Sebastian H. R. Wurst, Gregor Lamla, Florian Kohlmayer, Rainer Blaser, Dominik Schmelcher, Bernd Vögele, and Klaus A. Kuhn. Informatics and translational medical research: challenges and developments. *it - Information Technology*, 53(5):217–226, 2011.

[QL08] Bastian Quilitz and Ulf Leser. Querying distributed RDF Data sources with SPARQL. In *Proceedings of the 5th European Semantic Web Conference*, 2008.

[Qua07] Dennis Quan. Improving life sciences information retrieval using semantic web technology. *Briefings in Bioinformatics*, 8(3):172–182, May 2007.

[RCB$^+$07] Alan Ruttenberg, Tim Clark, William Bug, Matthias Samwald, Olivier Bodenreider, Helen Chen, Donald Doherty, Kerstin Forsberg, Yong Gao, Vipul Kashyap, June Kinoshita, Joanne Luciano, M Scott Marshall, Chimezie Ogbuji, Jonathan Rees, Susie Stephens, Gwendolyn T Wong, Elizabeth Wu, Davide Zaccagnini, Tonya Hongsermeier, Eric Neumann, Ivan Herman, and Kei-Hoi Cheung. Advancing translational research with the semantic web. *BMC Bioinformatics*, 8(Suppl 3):S2, 2007.

[RDF] RDF vocabulary description language 1.0: RDF schema. Website. Available online at `http://www.w3.org/TR/rdf-schema/`; visited on July 1st 2012.

[REJ] Rej. Website. Available online at `http://infinitesque.net/projects/Legere/components/Rej/`; visited on July 1st 2012.

[RGP06] Jesús Barrasa Rodriguez and Asunción Gómez-Pérez. Upgrading relational legacy data to the semantic web. In *Proceedings of the 15th International World Wide Web Conference*, 2006.

[RIF] RIF overview. Website. Available online at `http://www.w3.org/TR/rif-overview/`; visited on July 1st 2012.

[RPB$^+$08] D. M. Roden, J. M. Pulley, M. A. Basford, G. R. Bernard, E. W. Clayton, J. R. Balser, and D. R. Masys. Development of a large-scale de-identified DNA biobank to enable personalized medicine. *Clinical Pharmacology and Therapeutics*, 84(3):362–369, September 2008.

[RTCS10] Jessica Ross, Samson W. Tu, Simona Carini, and Ida Sim. Analysis of eligibility criteria complexity in clinical trials. *Proceedings of the AMIA Summits on Translational Science*, 2010:46–50, March 2010.

[RU93] Raghu Ramakrishnan and Jeffrey D. Ullman. A survey of research on deductive database systems. *Journal of Logic Programming*, 23:125–149, 1993.

[Sam01] Pierangela Samarati. Protecting respondents' identities in microdata release. *IEEE Transactions on Knowledge and Data Engineering*, 13(6):1010–1027, 2001.

[Sar10] Indra N. Sarkar. Biomedical informatics and translational medicine. *Journal of translational medicine*, 8:22, 2010.

[Sch09] Klaus-Benedikt Schultis. Extraktion von Informationen aus HL7-Nachrichtenströmen. Interdisciplinary project (IDP), Technische Universität München, 2009. Supervised by Fabian Praßer.

[SCT$^+$10] Ida Sim, Simona Carini, Samson Tu, Rob Wynden, Brad H. Pollock, Shamim A. Mollah, Davera Gabriel, Herbert K. Hagler, Richard H. Scheuermann, Harold P. Lehmann, Knut M. Wittkowski, Meredith Nahm, and Suzanne Bakken. The human studies database project: Federating human studies design data using the ontology of clinical research. *Proceedings of the 2010 AMIA Summits on Translational Science*, 2010:51–55, March 2010.

[SCY$^+$07] Andrew K. Smith, Kei-Hoi Cheung, Kevin Y. Yip, Martin Schultz, and Mark B. Gerstein. LinkHub: a semantic web system that facilitates cross-database queries and information retrieval in proteomics. *BMC Bioinformatics*, 8(Suppl 3):S5, 2007.

[SDB10] Marcos Antonio Vaz Salles, Jens Dittrich, and Lukas Blunschi. Intensional associations in dataspaces. In *Proceedings of the 26th International Conference on Data Engineering*, 2010.

[SGH$^+$11] Michael Schmidt, Olaf Görlitz, Peter Haase, Günter Ladwig, Andreas Schwarte, and Thanh Tran. FedBench: A benchmark suite for federated semantic data query processing. In *Proceedings of the 10th International Semantic Web Conference*, 2011.

[SGK$^+$08] Lefteris Sidirourgos, Romulo Goncalves, Martin L. Kersten, Niels Nes, and Stefan Manegold. Column-store support for rdf data management: not all swans are white. In *Proceedings of the 34th International Conference on Very Large Data Bases*, 2008.

[SHH$^+$11] Andreas Schwarte, Peter Haase, Katja Hose, Ralf Schenkel, and Michael Schmidt. FedX: Optimization techniques for federated query processing on linked data. In *Proceedings of the 10th International Semantic Web Conference*, 2011.

[SHLP09] Michael Schmidt, Thomas Hornung, Georg Lausen, and Christoph Pinkel. SP2Bench: A sparql performance benchmark. In *Proceedings of the 25th International Conference on Data Engineering*, 2009.

[SHX$^+$05] Sohrab Shah, Yong Huang, Tao Xu, Macaire Yuen, John Ling, and Francis B. F. Ouellette. Atlas - a data warehouse for integrative bioinformatics. *BMC Bioinformatics*, 6(1):34, 2005.

[SIL] The SILK project: Semantic inferencing on large knowledge. Website. Available online at `http://silk.semwebcentral.org/`; visited on July 1st 2012.

[SIM] Simbioms - home. Website. Available online at `http://simbioms.org/`; visited on July 1st 2012.

[SJB$^+$11] Matthias Samwald, Anja Jentzsch, Chritopher Bouton, Claus Stie Kallesøe, Egon Willighagen, Janos Hajagos, Scott M. Marshall, Eric Prud'hommeaux, Oktie Hassenzadeh, Elgar Pichler, and Susie Stephens. Linked open drug data for pharmaceutical research and development. *Journal of Cheminformatics*, 3:19, May 2011.

[SKBK01] Konrad Stocker, Donald Kossmann, Reinhard Braumandl, and Alfons Kemper. Integrating semi-join-reducers into state of the art query processors. In *Proceedings of the 17th International Conference on Data Engineering*, 2001.

[SKKP10] Sándor Szalma, Venkata Koka, Tatiana Khasanova, and Eric D. Perakslis. Effective knowledge management in translational medicine. *Journal of Translational Medicine*, 8(1):68, July 2010.

[SL90] Amit P. Sheth and James A. Larson. Federated database systems for managing distributed, heterogeneous, and autonomous databases. *ACM Computing Surveys*, 22(3):183–236, September 1990.

[SPA] SPARQL 1.1 federation extensions. Website. Available online at `http://www.w3.org/TR/sparql11-federated-query/`; visited on July 1st 2012.

[SPI] SPIN - SPARQL inferencing notation. Website. Available online at `http://spinrdf.org/`; visited on July 1st 2012.

[SR06] Julian Seidenberg and Alan L. Rector. Web ontology segmentation: analysis, classification and use. In *Proceedings of the 15th International World Wibe Web Conference*, 2006.

[SS98] Pierangela Samarati and Latanya Sweeney. Generalizing data to provide anonymity when disclosing information. In *Proceedings of the 17th ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems*, 1998.

[SS08] Simon Schenk and Steffen Staab. Networked graphs: a declarative mechanism for SPARQL rules, SPARQL views and RDF data integration on the web. In *Proceedings of the 17th International World Wide Web Conference*, 2008.

[Str11] Adrian Streitz. Übersicht über vorhandene Ansätzen zur Transformation relationaler Daten zum Zwecke der ontologiebasierten Integration. Interdisciplinary project (IDP), Technische Universität München, 2011. Supervised by Fabian Praßer.

[SVHB04] Heiner Stuckenschmidt, Richard Vdovjak, Geert-Jan Houben, and Jeen Broekstra. Index structures and algorithms for querying distributed RDF repositories. In *Proceedings of the 13th International World Wide Web Conference*, 2004.

[Swe97] Latanya Sweeney. Datafly: A system for providing anonymity in medical data. In *Proc of the 11th International Conference on Database Security*, 1997.

[Swe01] Latanya Sweeney. *Computational Disclosure Control - A Primer on Data Privacy Protection*. PhD thesis, Massachusetts Institute of Technology, 2001.

[Swe02] Latanya Sweeney. Achieving k-Anonymity privacy protection using generalization and suppression. *International Journal of Uncertainty, Fuzziness and Knowledge-Based Systems*, 10(5):571–588, 2002.

[TAJ$^+$10] Can Türker, Fuat Akal, Dieter Joho, Christian Panse, Simon Barkow-Oesterreicher, Hubert Rehrauer, and Ralph Schlapbach. B-Fabric: the swiss army knife for life sciences. In *Proceedings of the 13th International Conference on Extending Database Technology*, 2010.

[TOP] TopQuadrant | products | TopBraid composer. Website. Available online at `http://www.topquadrant.com/products/TB_Composer.html`; visited on July 1st 2012.

[TPC$^+$11] Samson W. Tu, Mor Peleg, Simona Carini, Michael Bobak, Jessica Ross, Daniel Rubin, and Ida Sim. A practical method for transforming free-text eligibility criteria into computable criteria. *Journal of Biomedical Informatics*, 44(2):239–250, April 2011.

[Tro11] Uwe Trottmann. Entwicklung einer Softwarebibliothek zur Transformation relationaler Daten in gerichtete Graphen. Interdisciplinary project (IDP), Technische Universität München, 2011. Supervised by Fabian Praßer.

[TWH09] Thanh Tran, Haofen Wang, and Peter Haase. Hermes: Data web search on a pay-as-you-go integration infrastructure. *Journal of Web Semantics*, 7(3):189–203, 2009.

[UNI] UniProt. Website. Available online at `http://www.uniprot.org/`; visited on July 1st 2012.

[Vaa11] Christian Vaas. Implementierung einer domanenspezifischen Benutzeroberfläche zur Transformation relationaler Daten in gerichtete Graphen. Interdisciplinary project (IDP), Technische Universität München, 2011. Supervised by Fabian Praßer.

[Van08] Jan P. Vandenbroucke. Observational research, randomised trials, and two views of medical science. *PLoS Medicine*, 5(3):e67, 2008.

[VBS+09] Teeradache Viangteeravat, Ian M. Brooks, Ebony J. Smith, Nicolas Furlotte, Somchan Vuthipadadon, Rebecca Reynolds, and Chanchai Singhanayok McDonald. Slim-prim: a biomedical informatics database to promote translational research. *Perspectives in Health Information Management*, 6:6, 2009.

[VBV+09] Teeradache Viangteeravat, Ian Brooks, Somchan Vuthipadadon, Eunice Huang, Ebony Smith, Ramin Homayouni, and Chanchai McDonald. Slim-prim: an integrated data system for clinical and translational research. *BMC Bioinformatics*, 10(Suppl 7):A11, 2009.

[VOI] Describing linked datasets with the VoID vocabulary. Website. Available online at `http://www.w3.org/TR/void/`; visited on July 1st 2012.

[vW12] Ulrich von Waldow. Eine vergleichende Übersicht über Semantic Reasoner für den Einsatz in einer inkrementellen Integrationslösung für biomedizinische Daten. Interdisciplinary project (IDP), Technische Universität München, 2012. Supervised by Fabian Praßer.

[WB05] Robert L. Wears and Marc Berg. Computer technology and clinical work: still waiting for Godot. *Journal of the American Medical Association*, 293(10):1261–1263, March 2005.

[WBHF03] Debra L. Weiner, Atul J. Butte, Patricia L. Hibberd, and Gary R. Fleisher. Computerized recruiting for clinical trials in real time. *Annals of Emergency Medicine*, 41(2):242–246, February 2003.

[Web12] Dominik Weber. Implementierung einer GUI zur Administration von Transformationsworkflows in einer ontologiebasierten Integrationslösung für biomedizinische Daten. Interdisciplinary project (IDP), Technische Universität München, 2012. Supervised by Fabian Praßer.

[WLL+07] Timo Weithöner, Thorsten Liebig, Marko Luther, Sebastian Böhm, Friedrich Henke, and Olaf Noppens. Real-world reasoning with owl. In *Proceedings of the 4th European Semantic Web Conferenc*, 2007.

[WLP+09] Sebastian H. R. Wurst, Gregor Lamla, Fabian Prasser, Alfons Kemper, and Klaus A. Kuhn. Einsatz von dataspaces für die inkrementelle informationsintegration in der medizin. In *GI-Jahrestagung*, LNI, 2009.

[WMM+09] Griffin Weber, Shawn N. Murphy, Andrew J. McMurry, Douglas MacFadden, Daniel J. Nigrin, Susanne Churchill, and Isaac S. Kohane. The shared health research information network (shrine): A prototype federated query tool for clinical data repositories. *Journal of the American Medical Informatics Association*, 16(5):624–630, 2009.

[Woo08] Steven H. Woolf. The meaning of translational research and why it matters. *Journal of the American Medical Association*, 299(2):211–213, January 2008.

[Wor07] caBIG Strategic Planning Workspace. The cancer biomedical informatics grid (caBIG): infrastructure and applications for a worldwide research community. *Studies in Health Technology and Informatics*, 129(Pt 1):330–334, 2007.

[Wur11] Sebastian H. R. Wurst. *Dataspace Integration in der Medizinischen Forschung*. PhD thesis, Technische Universität München, 2011.

[WVV$^+$01] H. Wache, T. Vögele, U. Visser, H. Stuckenschmidt, G. Schuster, H. Neumann, and S. Hübner. Ontology-based integration of information - a survey of existing approaches. In *Proceedings of the IJCAI-01 Workshop on Ontologies and Information Sharing*, 2001.

[WWS$^+$10] Rob Wynden, Mark G. Weiner, Ida Sim, Davera Gabriel, Marco Casale, Simona Carini, Shannon Hastings, David Ervin, Samson Tu, John H. Gennari, Nick Anderson, Ketty Mobed, Prakash Lakshminarayanan, Maggie Massary, and Russ J. Cucina. Ontology mapping and data discovery for the translational investigator. In *Proceedings of the 2010 AMIA Summits on Translational Science*, volume 2010, 2010.

[Wyn08] Rob Wynden. An alternative approach to integrated data repository design. Website, 2008. Available online at `https://www.ctsacentral.org/documents/bahdocs/NewApproachestoDataRepositoryArchitecture,RobWynden(UCSF).pdf`; visited on July 1st 2012.

[XC04] Huiyong Xiao and Isabel F. Cruz. RDF-based metadata management in peer-to-peer systems. In *Proceedings of the 2nd IST Workshop on Metadata Management in Grid and P2P Systems*, 2004.

[ZB06] Michael Zeller and Tom Barbaro. A face is exposed for aol searcher no. 4417749. The New York Times, August 2006.

[Zer05] Elias A Zerhouni. Translational and clinical science–time for a new vision. *The New England Journal of Medicine*, 353(15):1621–1623, October 2005.

[Zie12] Johannes Ziegltrum. Dynamisches Scheduling von Transformationsworkflows in einer ontologiebasierten Integrationslösung für biomedizinische Daten. Interdisciplinary project (IDP), Technische Universität München, 2012. Supervised by Fabian Praßer.

[ZMC$^+$11] Lei Zou, Jinghui Mo, Lei Chen, Tamer M. Özsu, and Dongyan Zhao. gStore: answering SPARQL queries via subgraph matching. In *Proceedings of the 37th International Conference on Very Large Data Bases*, 2011.

[ZRGD10] Eric Zapletal, Nicolas Rodon, Natalia Grabar, and Patrice Degoulet. Methodology of integration of a clinical data warehouse with a clinical information system: the HEGP case. *Studies in Health Technology and Informatics*, 160(Pt 1):193–197, 2010.