TU WIEN Informatics

Harmonization System for Data Analytics using Microservices

DIPLOMARBEIT

zur Erlangung des akademischen Grades

Diplom-Ingenieur

im Rahmen des Studiums

Software Engineering & Internet Computing

eingereicht von

Edzus Milasus, BSc
Matrikelnummer 01409784

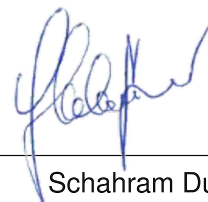
an der Fakultät für Informatik

der Technischen Universität Wien

Betreuung: Univ. Prof. Dr. Schahram Dustdar
Mitwirkung: Dr. Praveen Kumar Donta

Wien, 7. Mai 2024

Edzus Milasus



Schahram Dustdar

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.

Harmonization System for Data Analytics using Microservices

DIPLOMA THESIS

submitted in partial fulfillment of the requirements for the degree of

Diplom-Ingenieur

in

Software Engineering & Internet Computing

by

Edzus Milasus, BSc
Registration Number 01409784

to the Faculty of Informatics

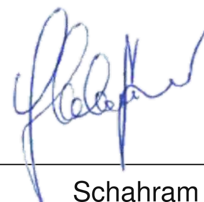
at the TU Wien

Advisor: Univ. Prof. Dr. Schahram Dustdar
Assistance: Dr. Praveen Kumar Donta

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

Vienna, 7th May, 2024

Edzus Milasus



Schahram Dustdar

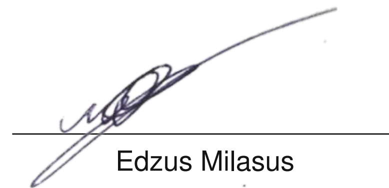
Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.

Erklärung zur Verfassung der Arbeit

Edzus Milasus, BSc

Hiermit erkläre ich, dass ich diese Arbeit selbständig verfasst habe, dass ich die verwendeten Quellen und Hilfsmittel vollständig angegeben habe und dass ich die Stellen der Arbeit – einschließlich Tabellen, Karten und Abbildungen – die anderen Werken oder dem Internet im Wortlaut oder dem Sinn nach entnommen sind, auf jeden Fall unter Angabe der Quelle als Entlehnung kenntlich gemacht habe.

Wien, 7. Mai 2024



Edzus Milasus

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.

Danksagung

Hiermit möchte ich mich bei allen Personen bedanken, die mich während meiner Masterarbeit und während meinem Masterstudium betreut und unterstützt haben.

Zuallererst gebührt ein großer Dank meinem Betreuer, Univ. Prof. Dr. Schahram Dustdar, und meinem Co-Betreuer, Praveen Kumar Donta, PhD. Für die Unterstützung, die konstruktive Kritik und Feedback möchte ich mich herzlich bedanken.

Ein besonderer Dank gebührt den beiden Gründern von Sustainista GmbH, Dr. Stefan Bauer und Alin Kalam MSc, MBA, für das Analysieren der Arbeit aus der Industriesicht, für die Bereitstellung von Testdatensätzen und die Zurverfügungstellung von einer vollständigen Programmier- und Deploymentumgebung.

Anschließend möchte ich mich bei meiner Familie und meinen Freunden herzlich bedanken. Vielen Dank für die immerwährende Unterstützung, die mir die Zeit und Energie gegeben hat, um dieses Masterstudium zu machen und die Masterarbeit zu schreiben.

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.

Acknowledgements

First and foremost, I would like to express my gratitude to the Technical University of Vienna for supporting me during my master's and the master thesis. Especially, my biggest thanks go to my supervisor Univ. Prof. Dr. Schahram Dustdar and my co-supervisor Praveen Kumar Donta, PhD, who generously provided knowledge, expertise and feedback during the contribution to this thesis.

Furthermore, I want to thank Sustainista GmbH and it's both co-founders Dr. Stefan Bauer and Alin Kalam MSc, MBA for providing the necessary support from the industry - giving input and feedback from the industry's perspective, providing data sets for this thesis and granting environment to develop, deploy and test the proof of concept implementation.

Special thanks go to my friends and family for helping me and supporting me every day. This support gave me the time and energy to work on this thesis and my master's in general.

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.

Kurzfassung

Verschiedene Industrien, wie Telekommunikation, Gesundheitswesen, Verkauf, Bankwesen, Marketing, Bildung, Agrikultur, Produktentwicklung, Energie, Versicherung u.a. produzieren jede Sekunde große Datenmengen. Algorithmen und verschiedene Programme sind notwendig, um diese Daten zu verarbeiten. Neue Entwicklungen in dem IoT Bereich produzieren immer höher werdende Datenmengen. Zusätzlich dazu, erhöhen verschiedene Neugeräte den Datenfluss ständig. Da nicht nur diese Geräte, sondern auch die Quellen der generierten Daten sehr unterschiedlich sein können, sind auch die damit entstehenden Daten sehr divers. Oft verschärft sich dieses Problem in Ausnahmesituationen, wenn nicht saubere Daten Echtzeitanalysen verlangsamen oder gar verhindern. Datenharmonisierung der unterschiedlichen Datensätze ist eine Methode, mit der diese Herausforderungen bewältigt werden können, da diese die Vergleichbarkeit der Daten erhöht. Über die Jahre sind viele Lösungen der Datenharmonizierung entstanden, jede mit den eigenen Vorteilen, aber auch Einschränkungen und Herausforderungen. Die Schwierigkeiten der Datenharmonisierung mit diesen Methoden existieren aber größtenteils weiterhin, da aus der Literatur sehr viele Methoden eine monolitische Natur haben. Um diese Probleme zu bewältigen, stellt diese Arbeit eine Microservices Architektur vor, die die Sammlung von Daten und deren Preprocessing mithilfe von Datenharmonisierungsmethoden ermöglicht. Die vorgestellte Microservices Architektur zur Datenharmonisierung erlaubt es, Daten über APIs oder von Dateien zu importieren. Hiermit wird die Flexibilität der Datensammlung erhöht. Weiters zeigen wir mithilfe von Speicher- und Rechenzeitanalysen, dass unser Ansatz in verschiedenen Bereichen effizient und effektiv arbeitet. Damit die Vorteile aufgezeigt werden können, haben wir verschiedene Szenarien erstellt. Auf der einen Seite haben diese das Aufräumen von Datensätzen, wie z.B. das Löschen von doppelten oder leeren Einträgen, beinhaltet, auf der anderen, haben diese sowohl öffentliche als auch private Datensätze berücksichtigt. Wir zeigen, dass das Importieren der Daten aus Datenbanken besser als der Import aus Dateien ist. Mit dem Ansatz der automatisierten Datenharmonisierung wird nicht nur der manuelle Aufwand während der Datenverarbeitung reduziert und die Qualität der Daten verbessert, sondern auch die Kosten gesenkt und die Datenintegrität erhöht.

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.

Abstract

Real-world applications such as healthcare, telecommunications, retail, law enforcement, banking, marketing, education, agriculture, new product development, energy and utilities, insurance, and urban planning produce massive amounts of data every second, and tools and algorithms are key to assessing this data. Recent advances in RFID such as IoT and sensing devices are also contributing a vast amount of data, and the amount of devices are also increasing the data generation continuously. Due to the diversity of devices and data gathering sources, it is highly incongruent, heterogeneous, and fragmented. Often, these issues are exacerbated in emergency situations when unclean data stalls real-time analyses. Data harmonization of different datasets is an increasingly common method of overcoming these data challenges by maximizing comparability. Over the years, multiple solutions have been developed for data harmonization with their own limitations, challenges and advantages. However, data harmonization complexity still exists in these methods and it's highly complex due to the monolithic nature of the majority of the methods in the literature. To overcome these issues, this thesis introduces a microservices architecture designed for automating data gathering and preprocessing using data harmonization methods. Our proposed microservices-based data harmonization architecture supports data importation from files or via APIs, facilitating flexibility in data sourcing. We analyze and prove our approach is efficient and effective in various aspects such as improving memory and computational time. To confirm the superiority we evaluated our method using various scenarios that included the efficacy of removing duplicate and empty entries in reducing memory consumption and preprocessing time over public and private datasets. Moreover, comparative analysis reveals that importing data from databases outperforms file-based imports. In addition to reducing manual overhead, our approach benefits from enhanced data quality through automated preprocessing, which can reduce costs and improve overall data integrity.

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.

Contents

Kurzfassung **xi**

Abstract **xiii**

Contents **xv**

1 Introduction **1**

2 Literature Review **5**

3 Microservice-based Harmonization **11**
3.1 System Overview 11
3.2 Data Collection and Access 13
3.3 Communication Layer 26
3.4 Data Preprocessing 34

4 Results **45**
4.1 Experimental setup 45
4.2 Effect on Memory and Computations 53

5 Conclusion **73**

List of Figures **75**

List of Tables **77**

Bibliography **79**

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

CHAPTER 1

Introduction

Data analytics and big data play an important role in today's digital world, spurred by social media interactions, searches, data collected from electronic devices (for example, wearable and IoT devices), data generated by artificial intelligence (AI) tools (for example, ChatGPT), weather data, and content consumption that produce exponential amounts of data [1]. In the age of big data, businesses are increasingly leveraging it to improve operational efficiency, gain insights into customer behavior, and drive innovation [2]–[4]. Businesses generate 1.7 megabytes of data per second on average, which is growing at a rate of 1.7 megabytes every second. The global big data market is expected to continue its robust growth trajectory, reaching $90 billion by 2025 and $103 billion by 2027[1]. Big data analytics is driving significant transformations in industries such as manufacturing, healthcare, and entertainment, with applications ranging from supply chain optimization to personalized content recommendations. In the future, the increase of internet-connected devices and the adoption of emerging technologies such as AI and machine learning (ML) will further increase the amount of data that can be analyzed [5]. Big data presents both opportunities and challenges for businesses in order to gain competitive advantage and make strategic decisions.

"Data is the new oil" was a speech given by Clive Humby in 2006, which emphasized on data being invaluable in it's raw state. After preprocessing and harmonizing, similarly as to oil refinment, data gains it's real value. By today's interpretation of the citation, not only the data needs to be refined to be valuable, but also the data itself is one of the most valuable resources in the world [6]. Our global emissions data were meticulously curated, with tonnes of carbon converted into tonnes of carbon dioxide (CO_2) using 3.664 as the conversion factor [7]. Kuwait's emission record for 1991 includes emissions from Kuwaiti oil fires, ensuring accuracy and completeness. Moreover, each country's share of global population has been meticulously calculated using a comprehensive population

[1] https://whatsthebigdata.com/big-data-statistics/

dataset gathered from reputable sources. The Global Carbon Budget dataset has also been utilized to determine each country's share of global CO2 emissions from flaring. This ensures a comprehensive and fair assessment of emissions [8].

For many years the amount of data has increased rapidly. Not only the amount of data, but also the diversity of data and the variety of sources from which data can be received have increased, e.g., sensor devices and IoT [9]. Thus the analysis of this data becomes increasingly cumbersome and requires more and more resources to process it properly. This data is highly incongruent, heterogeneous, and fragmented [10], [11]. Analyzing such heterogeneous and incongruent data is incredibly complex without preprocessing. Traditional preprocessing techniques are highly computational, requiring a lot of time for preprocessing and analysis [12], [13]. The data harmonization and analysis has to keep up with the rate at which this new data is generated [14]. This data needs to be gathered, harmonized, analyzed and provided to stakeholders or customers, depending on the industry. This is imperative, so that affluent learnings can be acquired from the data [15]. Such things can bolster the company, the government, and industries into investing in new technologies to advance their fields of interest. Harmonization is a crucial step in this learning process. Without it, multiple issues can arise. Harmonization cleans up data and prepares it for analysis. Data cleaning involves removing duplicate or empty entries from the dataset, for example.

Without the removal of these entries, the dataset might fill up with more unusable data rows. If this gets out of hand, the system might perform poorly. Even more alarming is that, if these datasets with incorrect entries are used in further data analysis, false outcomes might be produced. Moreover, different datasets might have different column names for the same values, such as dates - in one dataset, the year attribute might be called "year", while in another, it might be called "originatingFrom". During data analysis, these inconsistencies might cause thorny problems if there are too many of them. Having many different column names representing the same value can cause the overview to be lost easily. It may be helpful for those who work with these datasets to analyze the columns and rename them to the same column names [16]. In the event that more and more data is imported into the system, the previously mentioned problems might multiply rapidly. New data is generated continuously all over the world, so receiving new data into a data cluster is easy [17]. It is essential to properly preprocess data in order to avoid or at least mitigate problems that may arise. It is therefore necessary to consider automatic procedures since manual adaptations are time-consuming and ineffective. These harmonization systems can also pose a problem in terms of their setup [18]. In order for these systems to deliver large amounts of data in a short period of time in a reliable manner, their architectures must be carefully planned. It is necessary to provide a new harmonization architecture that can support people who use the system well, for example, reduce wait times, deliver data, which can be analyzed and integrated with other datasets.

This thesis proposes an architecture capable of automatically harmonizing different datasets. This involves gathering data from various public or private sources and

TU Bibliothek Your knowledge hub WIEN
Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.

importing data from file storage. After the data gathering phase is completed, the proposed architecture must harmonize the data in multiple dimensions, such as removing empty or duplicate entries, introducing new data columns for harmonization purposes, or renaming existing ones. All of this preprocessing needs to occur automatically without human interaction [19]. Once the harmonization process is finished, the architecture needs to provide a solution for saving the data in an easily accessible manner for stakeholders, customers, or other interested parties to access easily. After engineering the architecture, a prototype of the proposed architecture will be implemented, fulfilling all the main aspects including data gathering, data preprocessing, persisting data, and enabling data access [20]. As the prototype is completed, the objective is to import raw data into the system and execute the data preprocessing algorithms to ensure they work as intended. Furthermore, the implemented system needs to be evaluated with appropriate testing, including performance tests based on the elapsed time during a single data preprocessing run, and comparing memory requirements through before and after memory comparisons.

This study focuses on enhancing data analysis techniques, particularly optimizing existing data harmonization methods. In this context, we believe that dividing a monolithic architecture into multiple microservices can have benefits on the scalability and maintainability of the system [21]–[23]. If large systems with many different functionalities need to be put in place, microservices architecture, or more generally distributed systems, can be the go-to solution[24]. Additionally, concentrating on data and preprocessing, this thesis offers valuable insights that various industries and companies can leverage to automate their data harmonization processes. Such automation has the potential to yield significant economic benefits, especially if implemented in a manner that supports data analysts in streamlining data harmonization tasks.

In this thesis, a variety of methods were employed to overcome existing limitations. The proposed methodology is summarized in this thesis as follows.

- We propose a microservice-based harmonization system for data pre-processing. Further, we explore each microservice implementation in detail.
- We designed a single system with multiple microservices to access public and private databases separately without violating privacy constraints.
- We evaluated proposed microservices-based harmonization using publicly available Eurostat and CO2 emission datasets, and industry provided private datasets.
- We analyse various performance metrics such as memory and computational time under different scenarios.

The remaining chapters of this thesis are organized as follows. The chapter 2 of the thesis references literature reviews. In chapter 3 the proposed microservice-based harmonization architecture is discussed in detail. Chapter 4 discusses the detailed implementation of the architecture, data models and evaluation metrics with a detailed results analysis. Finally, we concluded our thesis with a future scope in Chapter 5.

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.

CHAPTER 2

Literature Review

This chapter provide various data gathering, analysis, accessing methods or architectures, available in the literature.

Over the years, various systems for gathering, analyzing, and accessing data have been developed in response to different needs within the software development community. Among these are federated systems, data gathering agents, and Internet of Things (IoT) systems, each serving distinct purposes that partially align with the objectives of the current thesis. A federated system, also known as a federated database system, is a collection of multiple separate databases. It is possible for these databases to have heterogeneous data or data formats but that is not always the case. Whenever a client or application requests data, the federated system can gather that information from these different sources and combine it as if it were locally stored. This approach enables the system to provide more accurate answers than if only one database was utilized independently [25].

Data gathering agents [26] work in a bottom-up approach to provide data to the users. Here the agents only access the information from agents beneath them in the hierarchy and/or from the raw data sources. These agents each are specialized in a specific domain, eg. geopolitical data, weather or transportation agents and are able to learn and improve on the made requests. In this work, the researches discussed, that it is important for such agents to fulfill the following attributes:

- *Modularity* - the agents should be responsible for one thing only and the combination of different source agents or databases allow the construction of more advanced agents with higher degree of information
- *Extensibility* - adding new agents should be simple. The process of exporting data from one agent to the next one should be easily doable.
- *Flexibility* - in regard to accessing different sources to combine the information or in case of outages the agents should be flexible enough to reroute to different

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek WIEN Your knowledge hub

sources or use cached information
- *Efficiency* - using parallelism, query optimization, caching and other tools to improve the performance of the agents in the sense of how fast a response is delivered after a request has been received
- *Adaptability* - possible changes in the data models should be easily doable

In the field of IoT devices there have been incentives into gathering and analyzing data. For this purpose multiple challenges have to be managed, eg. the amount of data received, the amount of devices sending data and the different protocols used [27]. Herefore, specific architectures have been thought of and developed. For example, architectures with multiple layers - the perception, the network and the application layer. Further sources use five or more layers [28], [29]. All of them focus on the communication in-between the IoT devices and between IoT devices and servers. Hardly any look into how the server internal architecture should look like to be more effective.

There are many different software architecture patterns thought of throughout the years. Each of them have different applications, advantages and disadvantages. Some of them are for example event-driven, microkernel, monolith and microservice software architectures. From these only microservice and monolith solutions would have been suitable for the system at hand. This is because the other two fulfill different objectives or the use case is not given in this project. For the event-driven architecture, events need to be used. But the project at hand does not have such event streams or events, which should be received and processed asynchronously [30]. The microkernel architecture is used for creating different packages of a product - having one core model and multiple possible add-ons, which can be added on top of the core model as needed [30]. Since the project at hand does not require different packaging, this pattern is not fitting for the system.

The choice for microservice and against monolith architecture was based on multiple reasons. First and foremost microservices are fine grained and they can be deployed separately[31]. This would allow future changes not to force a redeployment of the whole environment, but only the updated service. Furthermore, in a cloud environment, the services can be scaled separately. For example, as the amount of users of the system grow, the number of services experiencing high loads can be increased without scaling the other services[30]. For a monolith, only scaling of the whole system would be possible.

In this context also service-oriented architecture, short SOA, should be referred to. This software architecture pattern can be seen as one of the predecessors of the microservice architecture. SOA is complex, difficult to understand and implement, and in comparison to other architectures more expensive. [30] So, SOA was ruled out of the possible architecture types to implement. As the other architectural styles were ruled out due to different reasons, microservice system architecture was chosen as the system type to implement.

In general, microservices or microservice system architecture is a software system architecture, where multiple small services get implemented and deployed. These services should have multiple characteristics:[32]

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

- independently deployable
- loose coupling
- match business environment capabilities
- development is done by one small team

Since this is a small scale prototype, only the first two of these requirements can be fully fulfilled. As the industry partner's business around this project is in it's beginning phases, the business environment capabilities are known only to a limited extent. As for the fourth and final point - the development of a single small team for each of the services is not possible, since this is a master thesis project for a single person. In the future, though, the prototype can fulfill this requirement, as the project shifts fully into the industry partner's development environment.

Analyzing the existing microservice architectures and solutions was an important part of the literature research. In the literature reference [33] a data lake was setup for data storage and data analysis was included in an already existing microservices environment. Alongside using already existing technologies, like Apache Spark for data analytics, the thesis [33] suggests to implement multiple layers for the different steps - data ingestion, data storage, data analysis and access. This was also considered for the work at hand. In comparison to the work [33], the thesis at hand proposed a complete microservices architecture for the solution to be able to better influence the different parts of the system. Furthermore, in this thesis the data preprocessing is the final part of the process at hand. Further steps regarding data analysis are not a subject for this thesis. This way, the data preprocessing can be researched more in depth to find improved solutions on the one hand, and on the other - automatic preprocessing can be considered. Automatic preprocessing improves the performance of the preprocessing and reduces the time needed by a person preprocessing the data. Furthermore, also an architecture is proposed in the current work. If an architecture is implemented, more freedom can be achieved as in opposition to online environment with given tools.

The paper [34] proposed an integration of a microservices architecture into a medical hospital environment. This architecture should cover all of data management steps - gathering, storing, harmonizing and granting access to the data. The environment in paper [34] was analyzed and a specific solution with multiple services was proposed and implemented. Although a similar prototype was implemented, no in depth analysis of the harmonization was conducted. Furthermore, the proposed solution in source [34] is an environment specific solution with only medical data taken into consideration. Thus, it is not applicable for general use, as opposed to the solution proposed in this thesis.

A systematic literature review [35] discussed, what data harmonization techniques exist and how they are used. Data harmonization techniques with AI, natural language processing (NLP) and deep learning (DL) were looked at. Limitations to this systematic literature research [35] are, that here only theoretical work has been done, no implementations. Furthermore, only the analysis of data harmonization techniques has been made and no architecture solutions were discussed. When talking about other possibilities to harmonize data, this article [36] works with one of the newest technologies in the

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

field, where multiple datasets are merged. The author of [36] uses integrative data analysis to combine and analyze data from multiple source data sets. Integrative data analysis merges data sets by their common attributes, eg. Demographic information. The limitation to this work is on one hand, no microservices architecture was looked at and no general architecture, how to integrate this system, was analyzed. Furthermore, the datasets considered in this thesis are raw data sets without any existing links between one another. For the work in [36], datasets with similar data were considered.

This paper [37] handles the quality of the harmonized data and stresses how important it is, to have good quality data and proper harmonization, which doesn't lose any information during the harmonization. Furthermore, machine learning algorithms were included into the study [37] to see their impact on the data quality reduction. The limitations for the thesis [37] to this work are, that machine learning algorithm was incorporated, data leakage for the harmonized data sets was measured and compared for different harmonization methods. In this scientific paper [38] the authors conducted an analysis of data harmonization techniques in combination with how good machine learning algorithm perform after using data, that was harmonized with different methods. The paper [38] considers data harmonization techniques in comparison with machine learning. This is not done in the thesis at hand. Instead of looking for use cases between data harmonization and machine learning, a system for data preprocessing is proposed. The authors in paper [39] harmonized trade data with meta heuristic techniques. Although harmonizing trade data is also a part of general data harmonization, the paper [39] considers heuristic data harmonization, which is a different approach than proposed in this work.

Adhikari et al. proposed variable harmonization as a strategy to facilitate data pooling from heterogeneous datasets [40]. This paper illustrates how variables with differing measurement methods can be standardized to enable meaningful comparisons and analyses by describing specific harmonization strategies used in Canadian pregnancy cohort studies. *Kush et al.* provide a solution in [41] to address the challenges surrounding data sharing and interoperability in clinical research and healthcare, despite the recognized value of such practices by various stakeholders. It addresses various limitations, including common data elements (CDEs), such as lack of standardization, inconsistent terminology, and fragmented development processes, preventing comprehensive interoperability and data reuse. However, this method focus only on medical data and interoperability issues. *Torbati et al.* compared different data analysis methods for neuroimaging studies that included data from multiple scanners [42]. The study addresses technical variability related to image intensity scale differences and scanner effects. In this study, Torbati et al. examined methods that did not undergo any data transformation (RAW), that normalized intensity using RAVEL, that harmonized regionally using ComBat, and that combined these methods to reduce bias in neuroimaging data. *Kalter et al.* developed and implemented a flexible Data Harmonization Platform (DHP) designed to harmonize individual patient data (IPD) from multiple studies [43].

Firnkorn et al. a systematic and generic approach for harmonizing heterogeneous medical

TU Bibliothek Your knowledge hub WIEN
Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.

data within research networks, specifically focusing on lung cancer phenotype data in the context of the German Center for Lung Research [44]. An eight-step process for data harmonization is presented in the paper that integrates spreadsheets and Talend Open Studio to unify disparate data formats and definitions. This approach results in the creation of a common basic dataset consisting of 285 structured parameters for lung cancer research, implemented within an I2B2 research data warehouse. It simplifies the complex process of data harmonization, facilitating collaboration between researchers in the field of lung cancer and advancing research. *Papadimitroulas et al.* in [45] provides a comprehensive review and synthesis of key advancements and challenges in the intersection of radiomics, deep neural networks (DNNs), and explainable AI (XAI) within the context of modern radiation oncology. They also provide a framework for understanding the role of DNNs in analyzing medical images and uncovering hidden biomarkers by examining the evolution of artificial intelligence in radiation oncology and the increasing use of computational methods for personalized diagnosis and treatment precision. Additionally, the paper provides a comprehensive overview of the technical landscape, covering radiomic feature extraction, DNNs in image analysis, and major interpretability methods for explainable AI.

Fay et al. introduced SeaFlux in [46] which ensemble data product, which addresses the challenge of estimating the net flux of CO2 across the air-sea interface in a consistent and harmonized manner. With this data product, users can synthesize surface ocean CO2 observations from multiple sources into near-global coverage. SeaFlux ensembles address differences in spatial coverage and methodological inconsistencies by integrating six global observation-based mapping products and three wind products. In this work, surface ocean CO2 observations are harmonized and air-sea carbon fluxes are calculated using consistent inputs. This study emphasizes the importance of methodological consistency in estimating carbon fluxes by resulting in significant improvements in CO2 uptake for some products. *Tarazona et al.* proposed a comprehensive framework in [47] for multi-omics data analysis, which includes harmonized Figures of Merit (FoM) to assess quality and a MultiPower method to determine optimal sample sizes. Our work proposes a comprehensive framework for multi-omic data analysis, which includes harmonized Figures of Merit (FoM) to assess quality and a MultiPower method to determine optimal sample sizes. Introducing harmonized FoM fills a significant gap in the field in this paper, which provide standardized metrics for assessing the quality of multi-omic measurements across different omic technologies. MultiPower enables researchers to customize experimental designs based on specific data requirements, supporting different settings, data types, and sample sizes. Additionally, MultiML complements MultiPower by estimating sample sizes for machine learning classification problems using multi-omic data.

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.

CHAPTER 3

Microservice-based Harmonization

In this chapter, we discuss our proposed System Architecture for Harmonization (SAFH). Initially, data collection from multiple publicly accessible data sources (from different companies) has to be taken. All these imported data needs to be saved stored as (a) raw data (before preprocessing) and (b) preprocessed data. The preprocessed data needs to be available for the different customers and admins, and data analysis programs used internally.

This chapter will introduce the different parts of the SAFH system and provide a detailed discussions about them. An overview of the proposed SAFH system is provided using Figure. 3.1. As a part of SAFH system, we explain data collection, communication, and preprocessing stages in this chapter.

3.1 System Overview

As mentioned earlier, the system has to fulfill many different functionalities. Since the proposed system uses microservices architecture, for each of the main functionalities are consolidated into a separate microservice. For example, the general architecture (Figure 3.1) use following microservices:

- `Public Raw Data Extractor Microservice` (number 2) is used to extract the data from the different public data sources.
- `Raw Data Request Parameter Microservice` (number 3) has the functionality to save the request parameters necessary for requesting data from the public raw data sources.
- `Raw Data Microservice` (number 5) is necessary for managing raw public data.
- `Preprocessed Public Data Microservice` (number 7) manages data from publicly available sources after it has been preprocessed.

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

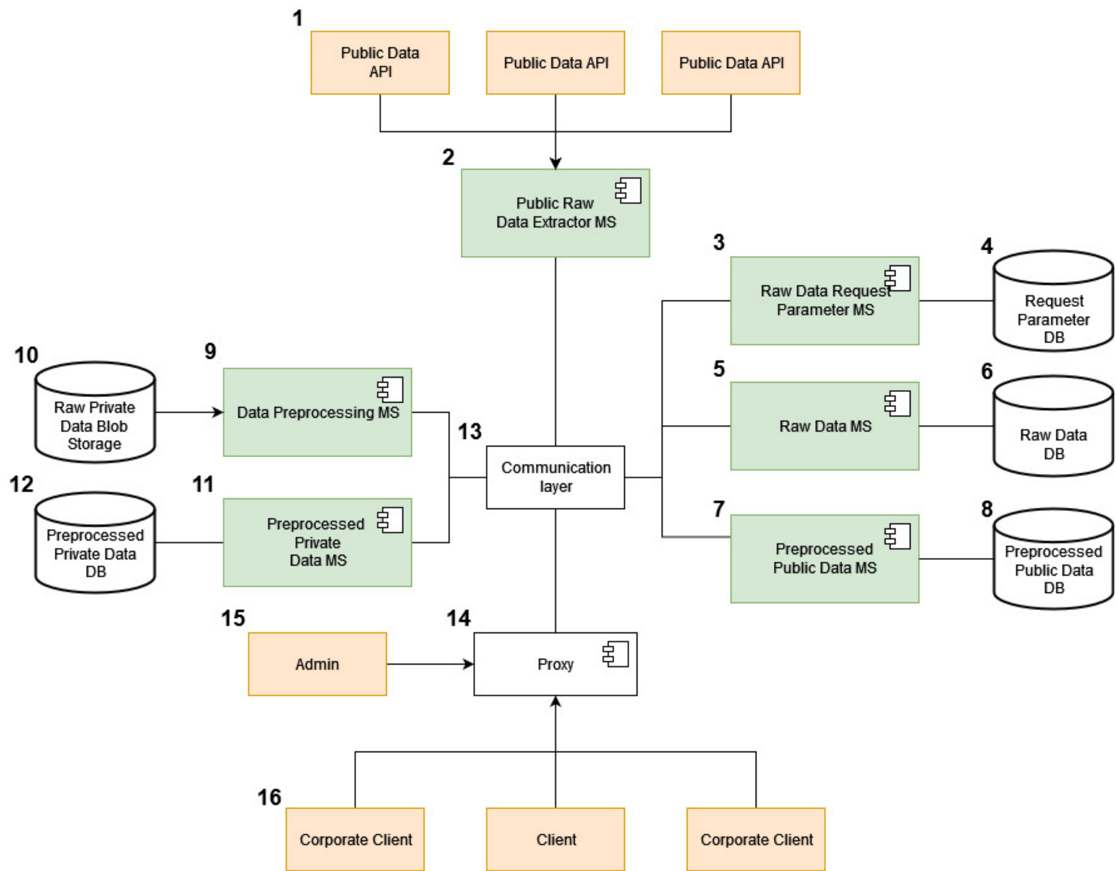


Figure 3.1: Overview of System Architecture for Harmonization

- `Data Preprocessing Microservice` (number 9) is responsible for preprocessing the gathered data.
- `Preprocessed Private Data Microservice` (number 11) is used to manage the private data after it has been preprocessed.

Subsequent sections of this chapter explain in detail about each microservice and their use. Also, these microservices persist and manage data, databases and data stores frequently, such as:

- `Request Parameter Database` (number 4) is required by the Raw Data Request Parameter Microservice.
- `Raw Public Data Database` (number 6) is used by the Raw Data Microservice.
- `Preprocessed Public Data Microservice` needs Preprocessed Public Data Databsae (number 8).
- `Raw Private Data Blob Storage` is used to save the raw private data files. These files are then imported by Data Preprocessing microservice.
- `Preprocessed Private Data Database` is required by Preprocessed Private

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek WIEN Your knowledge hub

Data Microservice.

Additionally to the data storage and the microservices, several other components are needed to fulfill harmonization. For example, inside the cloud system, a communication environment (to communicate with one another) between different microservices is necessary as shown in Figure 3.1 i.e., Communication Layer (number 13). A proxy (number 14) used by admins (number 15) and different clients (number 16) to interact with the system. The proxy is explained in detail in subsection 3.3.5. Also, publicly available data is considered as data source and it is acceded using Public Data APIs (number 1). Rest of this chapter discuss more details about data collection and access, communications, and processing in proposed Harmonization system and their microservices composition.

3.2 Data Collection and Access

Collecting data and enabling access to it is a core functionality of the proposed harmonization system. To ensure quality of service through systems functionalities, the data collection and access should be made easy, straight forward and repeatable as often as necessary. This section provide a detailed microservices composition for raw public data collection and storage followed by harmonized data. Further, we also provide details related to accessing it from outside the cloud system.

3.2.1 Public Raw Data Collecting and Storing

In this section, we focus on the critical aspect of data collection and storage within the proposed Harmonization architecture. We access various publicly available raw data sources and store the necessary data from these sources in the raw data databases.

Figure 3.2 shows the process of collection (import/extract) and storage of Raw Public data. The Admin (number 1) is the initiator of the raw data extraction. Admin can send an extraction request through Proxy (number 2) to the `Public Raw Data Extractor Microservice` (number 3). More details about the Proxy are discussed further in section 3.3. The Open Data API (number 4) is a public raw data source. The `Raw Data Request Parameter Microservice` (number 5) and the `Raw Data Microservice` (number 7) with the corresponding databases - Request Parameter Database (number 6) and Raw Data Database (number 8) - are both necessary microservices for data extraction and storage processes, and will be discussed in the upcoming subsections.

Raw Data Extractor Microservice

The `Raw Data Extractor Microservice` is used to extract public data from many different database sources. This service consists of multiple classes, which are depicted in Figure 3.3. In this figure, we can notice an external communication layer (number 1) and four classes interconnected with each other, ensuring different services, such as

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

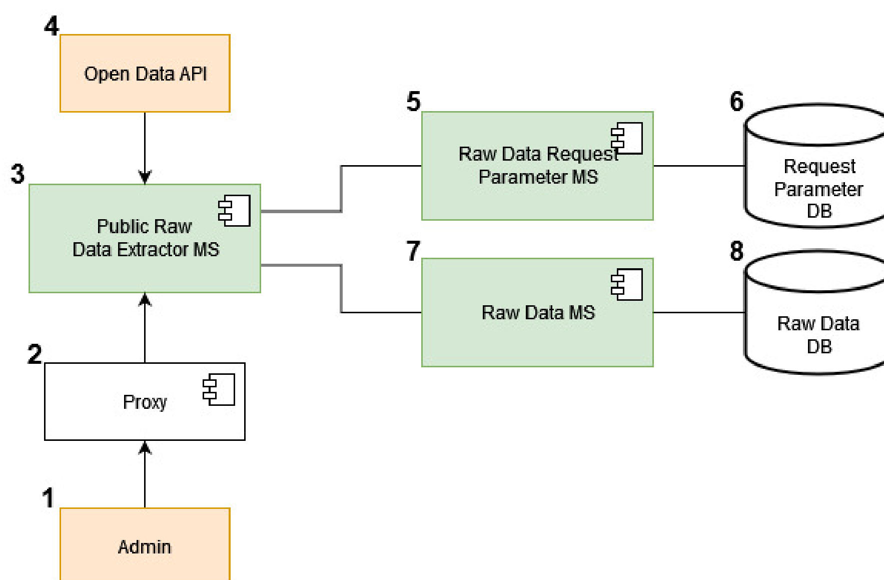


Figure 3.2: Public Raw Data Import Process Diagram

Communiator (number 2), RawDataExtractor (number 3), Manager (number 4) and RawDataParser (number 5) classes. Each class and their roles are described as follows:

Communicator Class (number 2) enable connection with other microservices in the cluster. Specially, it is useful to receive requests from other microservices and to get requests from the system users. The communicator class supports three functionalities through following functions:

- The `SetupCommunicator()` method serves as the initial setup phase, responsible for creating and initializing essential components such as communication protocols and configurations. For instance, if the system employs the HTTP protocol for communication, this method facilitates the creation and initialization of the "RestService" module, including configuration settings such as port properties. Additionally, it facilitates the setup of required endpoints to facilitate seamless communication. In cases where a message broker is utilized, this method also handles the establishment of subscriptions to various topics as needed.
- Upon receiving a request for data extraction from a designated source, additional details regarding request setup, such as headers, authentication methods, and specific URL paths, may be essential, which is taken care of by the `GetRequestParameters(...)` method. This essential information is stored within the `Raw Data Request Parameter Microservice`. Retrieval of this data necessitates the utilization of the `GetRequestParameters (...)` method. Through the use of a unique identifier (ID), the method retrieves the list of request parameters, forwarding them to the Manager class (number 4) for subsequent processing.

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

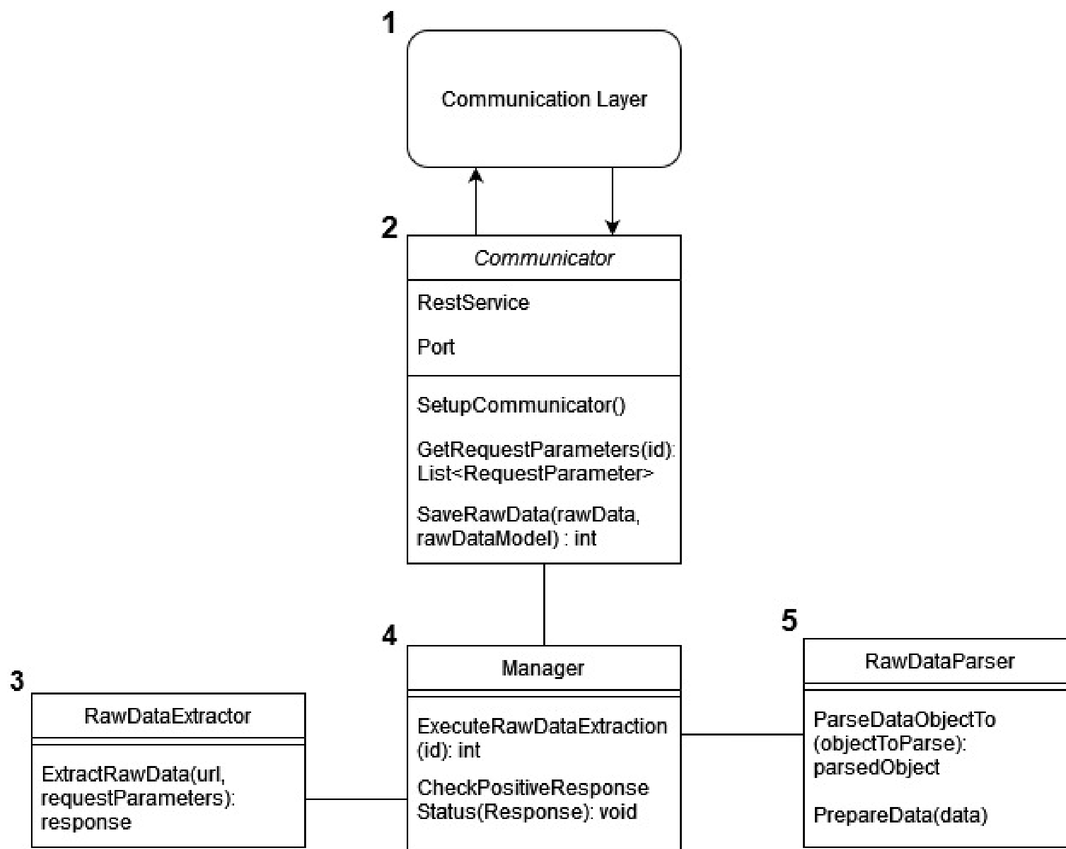


Figure 3.3: Raw Data Extractor Class Diagram

- `SaveRawData(...)` method is invoked following the completion of raw data extraction. This method orchestrates the storage process. Leveraging the RawDataModel, as elaborated in upcoming subsections, the extracted raw data is transmitted to the `Public Raw Data Microservice` for archival and storage.

RawDataExtractor Class A separate class `Raw Data Extractor` was introduced to retrieve data from a wide variety of public data sources (number 3). Comprised primarily of the `ExtractRawData(...)` method, this class orchestrates the formulation of requests to various external public data sources. There are numerous variations in request structures, URL paths, headers, responses, and data structures involved in data acquisition, which is due to the heterogeneity of these sources. Consequently, a meticulous analysis of each connected data source is crucial. It is essential that the components interfacing with raw public data within a microservice system are highly adaptable in order to accommodate this diversity. Thus, the recommended strategy involves creating a dedicated RawDataExtractor class, possibly designed as an interface with multiple implementations in the future.
Raw Data Extractor's limited scale results in a single method: ExtractRawData(...).

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

In this method, URLs and specific request parameters are expertly crafted, responses are retrieved, and they are seamlessly transmitted to the Manager class for processing.

Manager Class (number 4) is used as the central orchestrator within this microservice, tasked with the reception, transmission, and parsing of requests and data. Its functionality spans multiple key tasks, each meticulously executed to ensure an uninterrupted operation. Initially, the `Manager class` receives a data extraction request from the `Communicator` class (number 2). Subsequently, upon receipt of the initial request, the Manager class solicits additional request parameters specific to the data extraction process. The Manager class retrieves the required parameters by calling the `GetRequestParameter(...)` method from the Communicator class, as previously described.

In the subsequent phase, the `Manager class` interfaces with the `RawData Extractor` class (3), using the acquired request parameters, to begin capturing data from the specified source. Manager class executes initial adaptations with assistance from `RawDataParser` class (number 5) after receiving the `RawData Extractor` class' response. Through the `Communication Layer`, the Manager class facilitates the preparation and storage of raw data and returns the execution result to the originating caller. Through the entire execution cycle, the Manager class verifies the positivity of responses from various services and external sources. `CheckPositiveResponseStatus(...)` executes this validation process, which ensures that any encountered errors are promptly identified and relayed to the initial caller. When an error occurs, execution halts at the point of occurrence, and the relevant error message is relayed.

Within the Manager Class, the culmination of all previously outlined steps is encapsulated within the `ExecuteRawDataExtraction(...)` method. This method serves as the orchestrator for the entire raw data extraction process. The pseudocode for this method is delineated in Algorithm 3.11. Commencing with the display of the request parameter entry ID on line one, subsequent steps involve the reception and validation of request information from the `Request Parameter Service` on lines two to five, ensuring error-free processing. Lines six and seven initiate raw data extraction, followed by lines eight to ten handling errors meticulously. Upon successful extraction, the obtained raw data is dispatched to the `Raw Data Microservice` for storage on line eleven. After a final error check on lines 12 to 14, the raw data extraction process is completed by providing the resultant response to the caller on line 15.

Raw Data Parser Class In the microservices ecosystem, data originating from diverse sources often arrives in varying formats, requiring adaptations to ensure uniformity and compatibility. In the `Raw Data Parser class`, structural adjustments are made to align the received data with the standardized format used by other microservices in the cloud system. The prototype proposed the JSON format as a result of the variety of possible data types. The decision was influenced by widespread adoption, ease of integration, and compatibility with a variety

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

Algorithm 3.1: Execute Raw Data Extraction method

```
1  id
2  requestInformation ← getRequestParameters(id)
3  if requestInformation = error then
4  |   return error
5  end
6  rawData ← extractRawDataJSON(requestInformation.url,
7  requestInformation.requestParameters)
8  if rawData = error then
9  |   return error
10 end
11 savedData ← saveRawData(rawData, requestInformation.rawDataModel)
12 if savedData = error then
13 |   return error
14 end
15 return savedData
```

of programming languages. Microservice environments require simplicity and versatility, and technology stacks may change over time. This class is composed using `ParseDataObjectTo(...)` and `PrepareData(...)` methods.
In the first method, `ParseDataObjectTo(...)`, raw data is converted into the designated JSON format. The *PrepareData(...)* method then harmonizes the JSON structure to conform to other microservices' data structures. Integrating the internal cloud system seamlessly requires correcting discrepancies such as swapping data columns and rows. `PrepareData(...)` also performs basic data cleansing to enhance data quality. The dataset is streamlined and optimized by removing extraneous information such as timestamps and text descriptions. The elimination of empty data rows also mitigates potential distortions and enhances dataset integrity, which ultimately facilitates more accurate analysis. It is essential to perform this initial preprocessing step in order to refine the data, improve its quality, and simplify subsequent processing steps. Microservices are built on seamless integration and meaningful analysis of data that is cleansed at a basic level and standardized at this point.

Raw Data Request Parameter Microservice

The `Raw Data Request Parameter Microservice` serves as the central repository for storing and managing request parameters pertinent to various public raw data sources. To effectively handle this complexity, the Raw Data Request Parameter Microservice is proposed, facilitating streamlined management of diverse parameters. The class diagram of this service is depicted in Figure 3.4, showcasing its integral role within the architecture. `Communication layer` (number 1) is responsible for orchestrating

TU Bibliothek WIEN Your knowledge hub
Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.

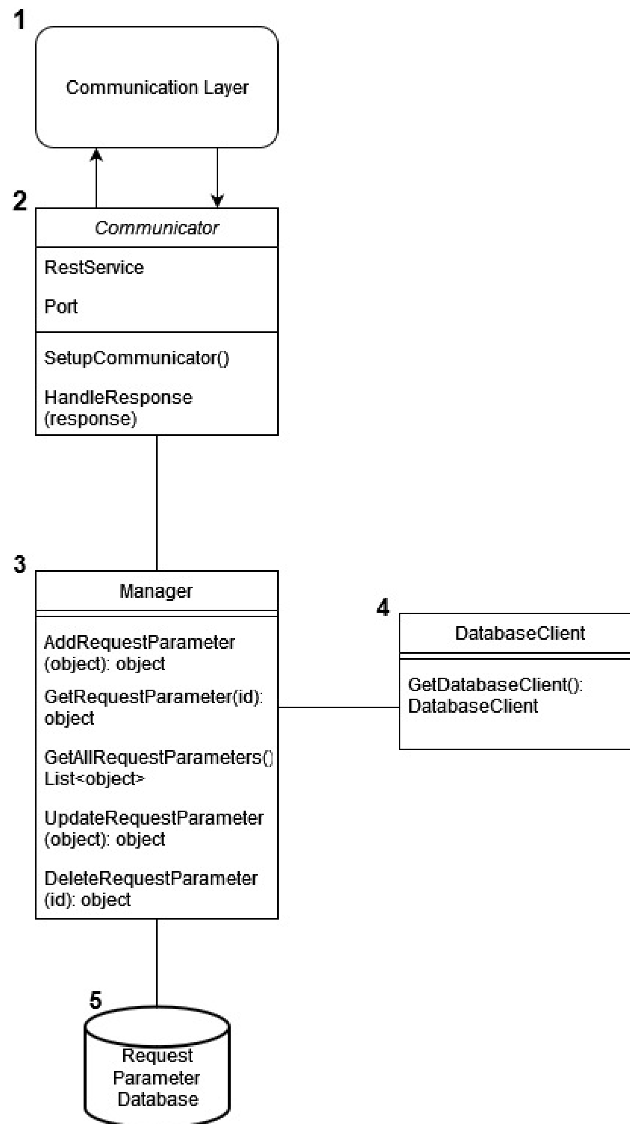


Figure 3.4: Raw Data Request Parameter Microservice Class Diagram

seamless communication between the internal cloud infrastructure and the users. A variety of components, such as `Communicator` (number 2), `Manager` (number 3) and `Database Client` (number 4), play a crucial role in managing and retrieving parameters. There will be further discussion of these components in subsequent subsections, culminating in an examination of the `Request Parameter Database` (number 5).

Communicator Class (number 2) functions as the central interface facilitating communication between the microservice and the internal cloud structure. Initially, the microservice must be set up with endpoints and services. A Rest endpoint

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek, WIEN Your knowledge hub

service is configured and the requisite endpoints are associated with it using the `SetupCommunicator()` method. These diverse endpoints support requests that involve creating, retrieving, updating, and deleting request data and parameter entries within the database. The Communicator class delegated these requests to the Manager class for further processing, maintaining a clear separation of concerns. After processing by the Manager class, the outcomes are relayed back to the Communicator class for delivery to the requester. In the event of errors, the `HandleResponse(...)` method facilitates the exchange of error messages and/or codes to enhance readability for the requester. At this stage, centralizing error handling reduces maintenance costs and simplifies microservice complexity. Additionally, ensuring a coherent communication workflow is facilitated by meticulously assigning HTTP error codes.

Manager Class act as a mediator between `Communicator`, `Request Parameter Database` (number 5) and `DatabaseClient` (number 4). This means, the requests received by Communicator class are passed on to the Manager class. After doing sanity checks, Manager class sends these requests to the `Request Parameter Database`. Following is a list of all the proposed requests for the database including their functionality descriptions:

- `AddRequestParameter(...)` – used to introduce a new request parameter to the database.
- `GetRequestParameter(id)` – used to get a request parameter based on an id.
- `GetAllRequestParameters()` – utilized for receiving all existing request parameters from the database.
- `UpdateRequestParameter(...)` – used for updating a request parameter.
- `DeleteRequestParameter(id)` – this method removes a request parameter from the database based on the id specified.

After these requests have been processed by the Request Parameter Database, the response is sent back to the `Manager class`. Subsequently, the `Manager class` conducts an error check on the response. If no errors are detected, the response is relayed back to the requester via the `Communicator class`. However, in the event of an error, the response is adjusted accordingly. This service relies heavily on the Manager class for incorporating essential business logic. This entails tasks such as verifying if specific integer values fall within defined ranges (e.g., if they are positive) or confirming the completeness of required fields, among other necessary checks. In contrast, the response received from the `Request Parameter Database` may require error checking and handling, along with possible data format adjustments. As the business logic components expand, the database communication might be segregated into a separate class. In this way, only business logic would be included in the `Manager class`. Nevertheless, given

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

the current prototype's scope, where extensive business logic implementation is not present, consolidating business logic and database requests into one class was sufficient.

Database Client Class (number 4) encapsulates this middleware functionality to interface with the database, and it is responsible for instantiating the specific database client and establishing all necessary connections to the database. Once the database client is instantiated, it can be retrieved by invoking the `GetDatabaseClient()` method. Typically, this method is called by the Manager class, which uses the database client to execute the database requests described above. The external Database Client class facilitates seamless interchangeability by allowing it to be substituted for alternative implementations or used concurrently with multiple implementations.

Request Parameter Database Raw Data Request Parameter Database (number 5) is used to save the request parameters required to make the HTTP calls necessary to request the raw data. To do so, the database has the following data structure:

- *id:* Int – unique identification number of the data source
- *name:* String? – an optional name for the raw data source
- *description:* String? – an optional description of the raw data source
- *url:* unique String – the full path to the external raw data source, from which the data should be extracted. This URL element is unique, so that no multiple copies of the same fully qualified path can be created. This was chosen this way, because each data source should only be equivalent to a single raw data model explained further down.
- *requestParameters* – all of the necessary additional request parameters. These could be additional headers, request type or others.
- *rawDataModel* – this entry corresponds to the raw data model name inside the `Public Raw Data Database` for this data source. It is used, so that the raw data can be inserted into the correct database table inside the `Public Raw Data Database`.

Public Raw Data Microservice

The `Public Raw Data Microservice` manages storage, updating, and access to data extracted from public sources. Once the data is structured according to the database schema managed by this microservice, it is transmitted to the service. Additionally, if a customer requires access to specific persisted raw data, the `Public Raw Data Microservice` is accessed to retrieve the requested data.

Similarly to the previously introduced `Raw Data Request Parameter` microservice, the `Public Raw Data Microservice` consists of three class and a communication layer. This structure can be seen in the following Figure 3.5. The communication layer (number 1) is, similarly to the other microservices, the connection point to the internal cloud environment. The classes - `Communicator` (number 2), `Manager` (number 3) and `Database Client` (number 4) - are explained next.

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

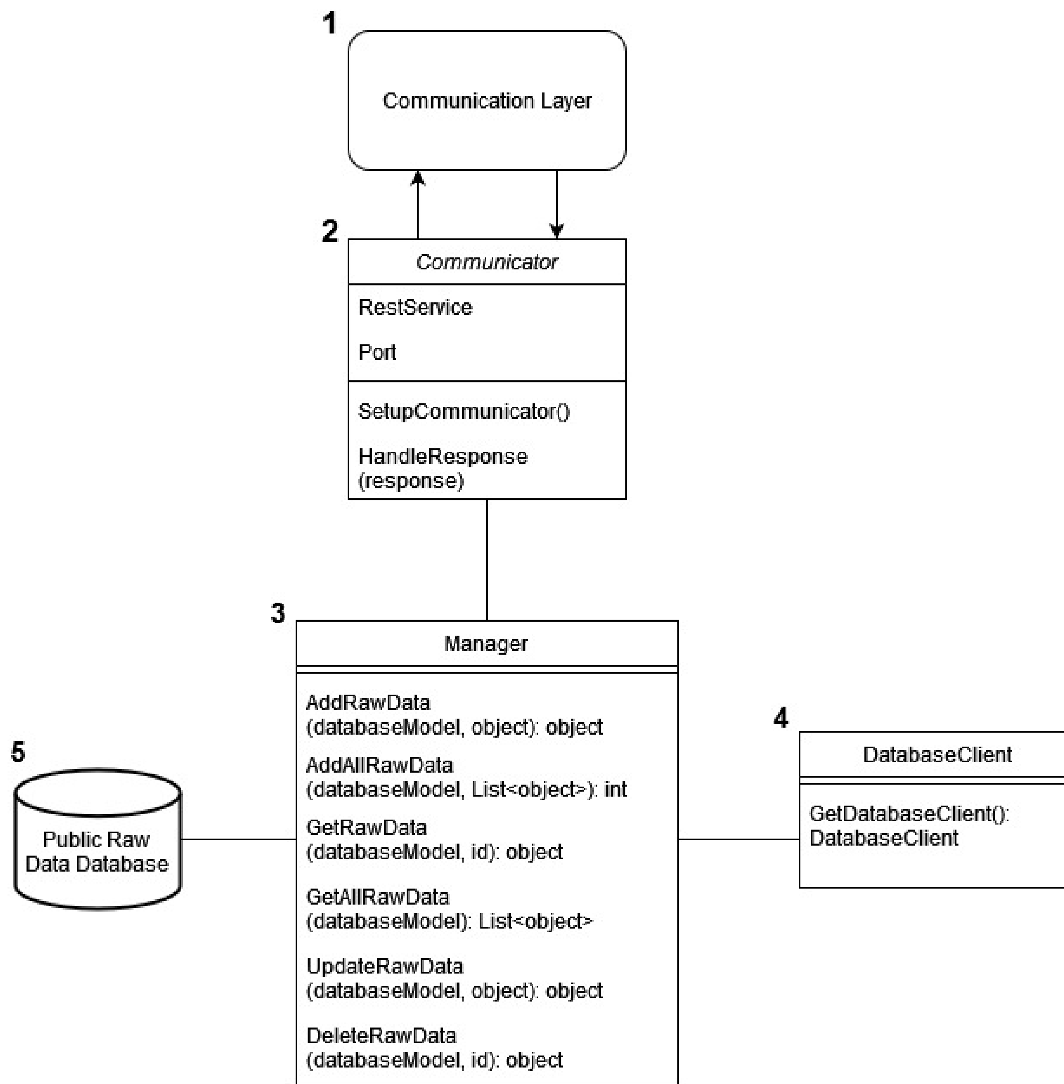


Figure 3.5: Public Raw Data Microservice Class Diagram

Communicator Class The Communicator class (number 2) works in the same manner as the Communicator class explained for the `Raw Data Request Parameter Microservice`. The only difference is, that there is an extra endpoint, `AddAll RawData(...)`, which allows to add a list of objects to the specified raw data table.

Manager Class within the `Public Raw Data Microservice` performs a similar function to the `Raw Data Request Parameter Microservice`. In general, its functionality is similar, acting as an intermediary between microservices and databases. However, closer examination reveals some notable differences.
Firstly, the method names differ. Methods include `RawData` instead of "Request-

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

Parameter" given the focus on raw data. For instance, `AddRawData(...)` is used to add a new raw data entry instead of `AddRequestParameter(...)`. In a similar manner, `GetRawData(...)` is employed to retrieve a single raw data entry from the database, and other methods follow the same pattern. Secondly, an additional method, `AddAllRawData(...)`, is introduced. The method facilitates the addition of new database entries based on raw data objects. The method inserts raw data objects into the database, and returns the number of entries. Lastly, a new method parameter, `databaseModel`, is necessitated for invoking these methods. This parameter refers to two aspects of the database. Primarily, it denotes the database model utilized for the received data, defining the fields, their quantity, and their types. Additionally, the database model corresponds to the name of the database table where the data is stored. By leveraging the `databaseModel` parameter, callers can dynamically interact with various raw data tables.

Database Client Class has similar functionalities as the `Database Client Class` inside the `Request Parameter Microservice`.

Public Raw Data Database The precise data models inside the `Public Raw Data Database` (number 5) are depending on the raw data, which is saved inside the database, because out of this data, the table columns can be created. A general example, which defines only the most basic requirements, is the following:

- *id: Int* – unique identification number for the entry. Since there could be many different types of data sources with different structures, it is difficult to exctract a general identification for all the possible sources. Thus, this id is an automatically incrementing number for each data row in an arbitrary raw data source table.
- *data: any* – any number of data columns with an arbitrary data type matching the data saved.

Figure 3.6 shows the public raw data processing and storing preprocessed Public data. The data is moved from Raw Public Data database (number 1) over the Raw Public Data microservice (number 2) to Data Preprocessing microservice (number 3). After the preprocessing has been finished, it is sent on to the Preprocessed Public Data microservice

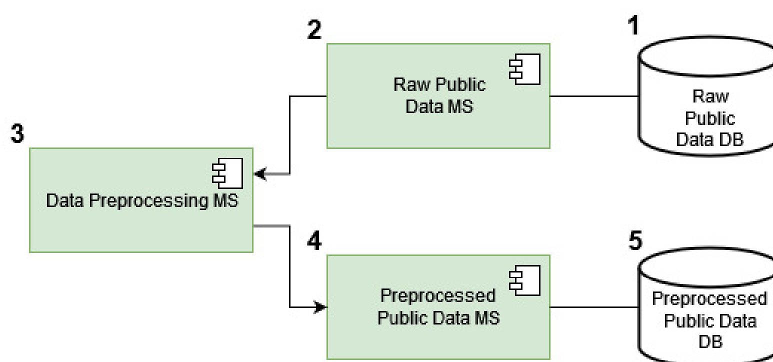


Figure 3.6: Public Raw Data Preprocessing working Process

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

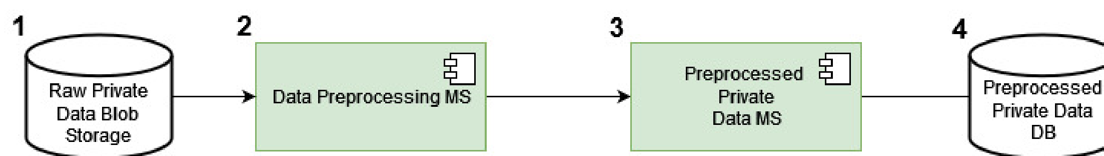


Figure 3.7: Private Raw Data Preprocessing

(number 4) and finally persisted inside the Preprocessed Public Data database (number 5).

3.2.2 Private Raw Data Storing

Data import into a cloud system can occur through API access or by transferring data via files, particularly suitable for private data. Blob storage, designed for storing unstructured data within cloud environments, facilitates this process. Raw Private Data Blob Storage serves this purpose, housing data files for subsequent importation, preprocessing by the Data Preprocessing Microservice, and onward transmission to the Preprocessed Private Data Microservice. While Excel format was utilized in this prototype, future implementations may incorporate CSV, JSON, or other formats.

Figure 3.7 shows Private raw data preprocessing, and it shows the data flow from the Raw Private Data Blob Storage (number 1) to the Data Preprocessing Microservice (number 2). After the data preprocessing is finished, the data is sent to the Preprocessed Private Data Microservice (number 3) and stored in the Preprocessed Private Data databse (number 4).

3.2.3 Preprocessed Public Data Storing

Preprocessed Public Data Microservice is responsible for the data harmonized/preprocessed by the `Data Preprocessing Microservice`. That means, this microservice saves the data, capable to return, update and, if necessary, delete it.

In the figure 3.8 structure of class `Preprocessed Public Data Microservice` is depicted. This microservice's structure is almost identical to `Public Raw Data Microservice`. One of the minor differences between these two microservices is the labeling of the methods inside the `Manager class`. The labeling `RawData` is used here as `PreprocessedData`. Another difference in this setup is the `Public Preprocessed Data Database` data models. In addition to the columns introduced in the raw data sets, additional columns can be introduced during the harmonization process, thus altering the data model. The new data structure includes the old data in addition to some new columns and might look as follows:

- *id: Int* – unique identification number for the entry. This should be the same as for the raw data.
- *data: any* – any number of data columns with an arbitrary data type matching the data saved.

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

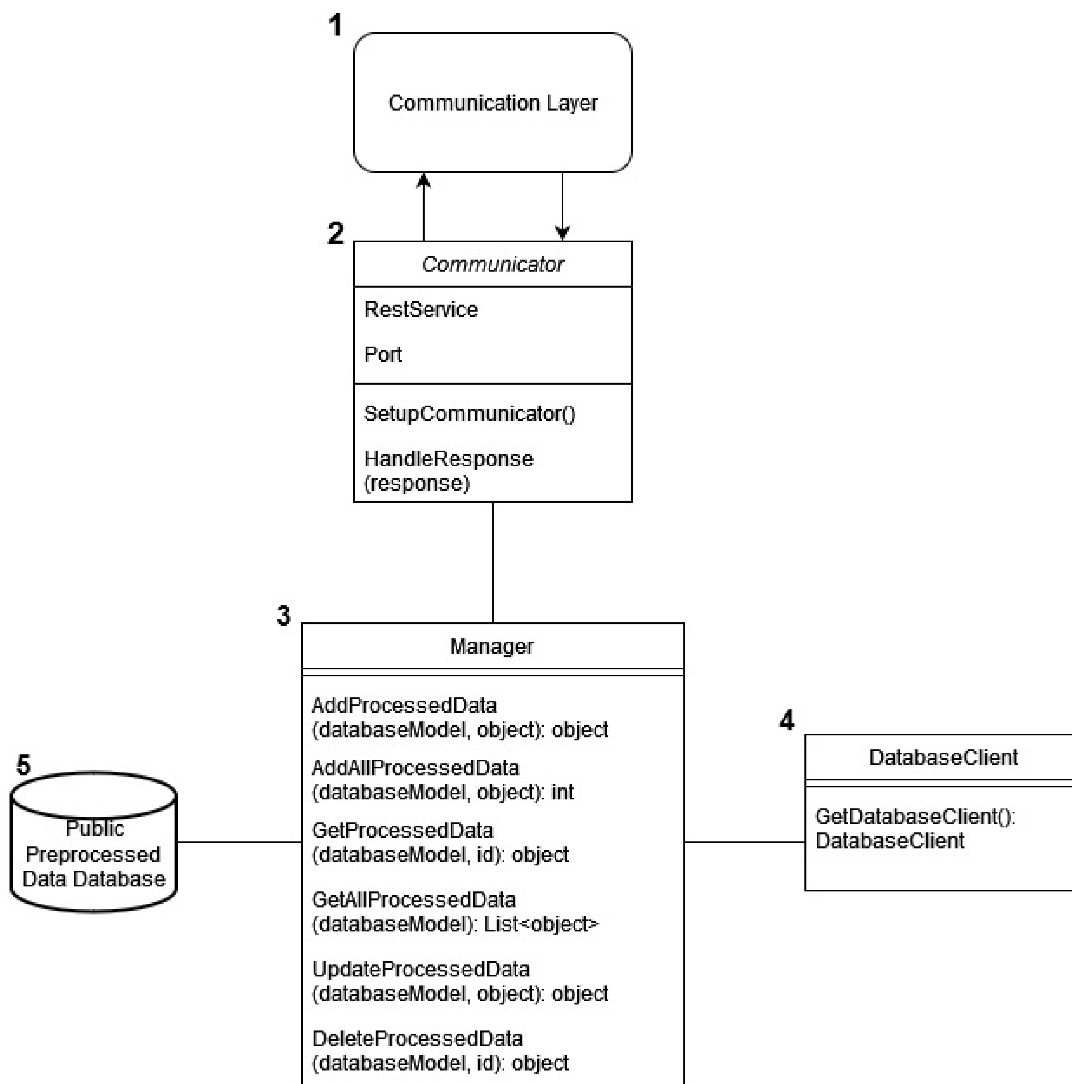


Figure 3.8: Preprocessed Public Data Microservice Class Diagram

- *harmonizedDataColumns: any* – table columns used for harmonizing data. They can have an arbitrary data type, depending on the data saved. For the harmonized data to be comparable, the harmonizedDataColumns need to apply to all the harmonized data. Examples for these columns could be year, country, industry etc.

3.2.4 Preprocessed Private Data Storing

The `Preprocessed Private Data Microservice` is dedicated to overseeing the private data preprocessed by the Data Preprocessing Microservice. Similarly to the previously discussed microservices that manage data, this service is capable of adding,

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

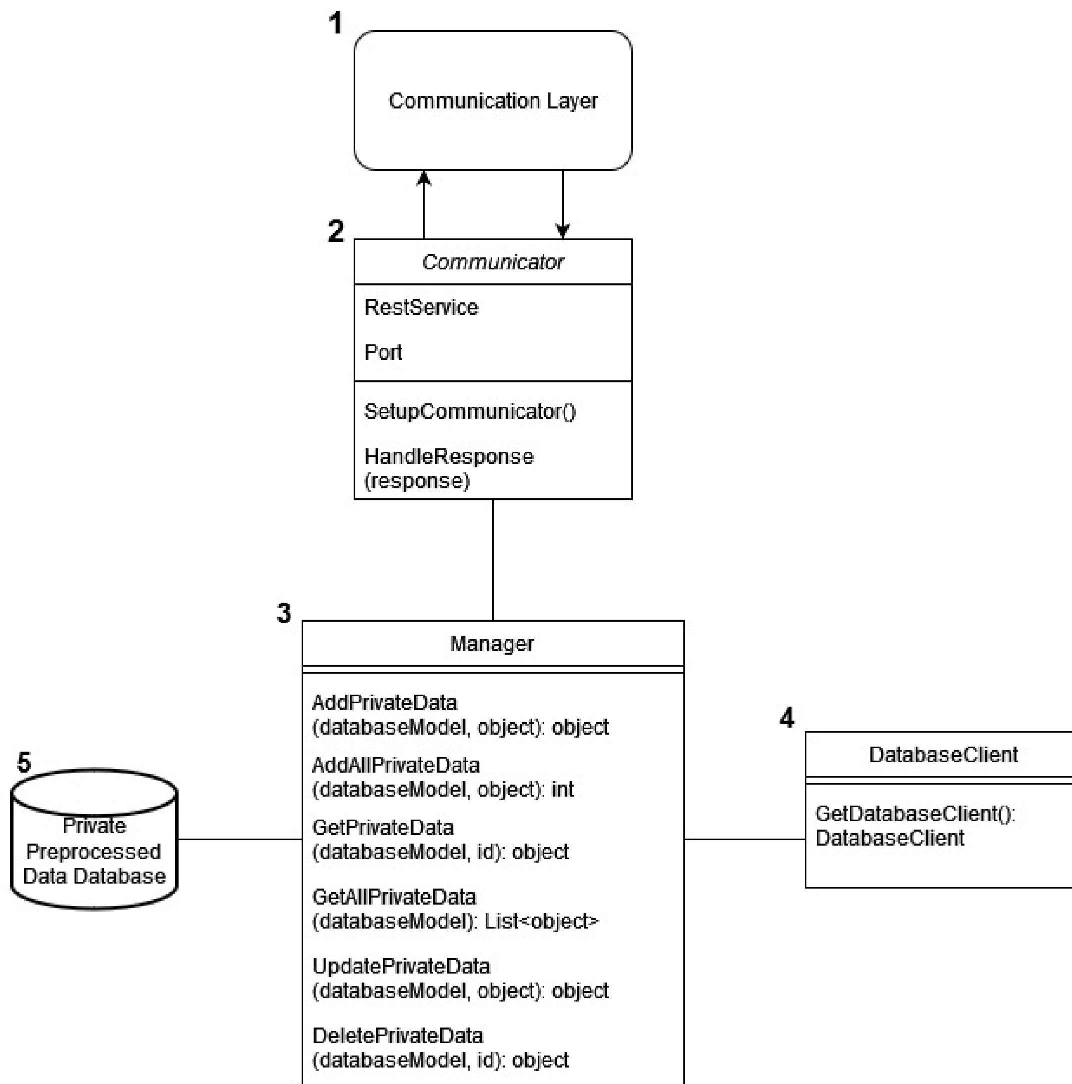


Figure 3.9: Preprocessed Private Data Microservice Class Diagram

retrieving, updating, and deleting preprocessed private data. The harmonization of the data also allows for various analyses of the data, including comparisons between private data and preprocessed public data, as well as comparisons between private data from different sources.

The class structure of the `Preprocessed Private Data Microservice` is depicted in Figure 3.9. The main difference between this microservice and `Public Raw Data Microservice` lies in their method names. In the Preprocessed Private Data Microservice, methods are labeled with `PrivateData` instead of `RawData`. Moreover, the shared objective of data harmonization persists - ensuring data comparability across disparate databases, thereby facilitating new findings.

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek WIEN Your knowledge hub

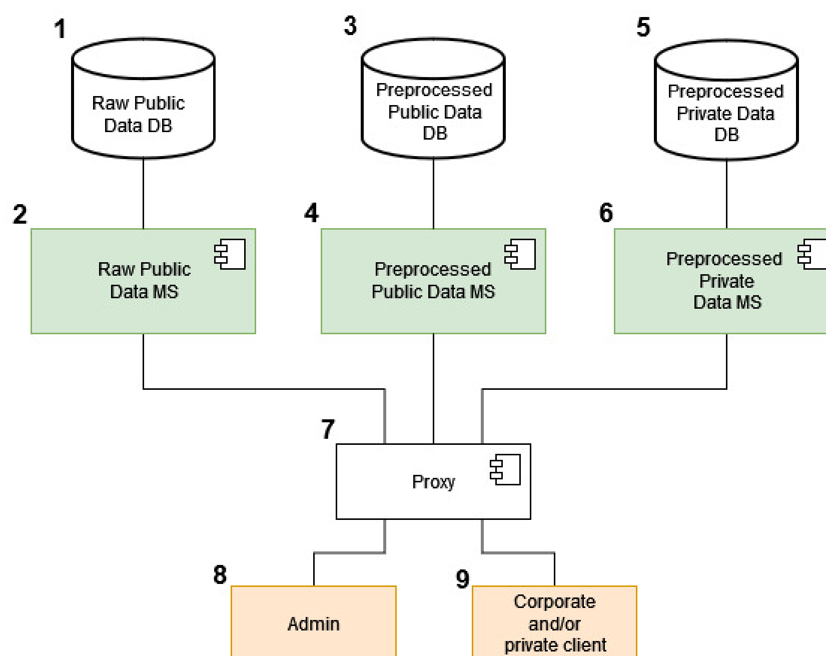


Figure 3.10: Proposed Data Access Process

3.2.5 Data Access

Accessing data is essential for comparing different datasets and discovering knowledge through data analysis. Figure 3.10 illustrates the environment for accessing system data. Communication channels between the databases for various data sources (1, 3, and 5) and their corresponding microservices (2, 4, and 6) have been delineated in the preceding subsections. The proxy (7) serves as the public access point to the cloud system, as elucidated in the communications section 3.3. Users, categorized as Admin (8) and clients (9), interact with the system to extract data. Notably, this prototype lacks authentication restrictions; however, future implementations will necessitate the implementation of access restrictions. These restrictions must delineate permissions to access specific data tables; for instance, data from one private source should not be accessible to another public or private client.

3.3 Communication Layer

Microservices are distributed in a cloud system, so effective communication is necessary to ensure seamless operation. It is essential for microservices to exchange information, transmit requests, and receive responses together in order to function cohesively. Therefore, the proposed harmonization system relies heavily on the communication layer. There are several communication patterns in the literature [48], among which messaging [49] and remote procedure invocation (RPI) [50] are noteworthy and suitable for the

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

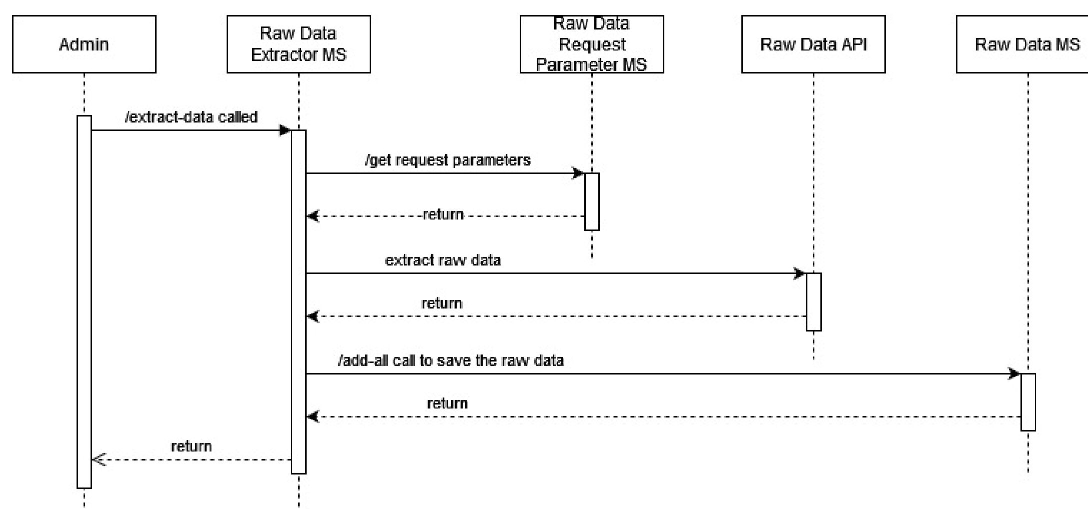


Figure 3.11: Sequence Diagram for data extraction

proposed harmonization system. In the proposed system, RPI was adopted, with the REST protocol serving as the chosen RPI mechanism. In the context of the REST protocol, microservice endpoints are imperative.

3.3.1 Extract Raw Public Data Communication Paths

Accessing raw data through public APIs is a complex process within the proposed cloud system. We visualize the entire process and understand the required endpoints and sequence of calls to the microservices by referring to the Sequence Diagram, shown in Figure 3.11.

From Figure 3.11 we notice that the process begins with the administrator invoking the "/extract-data" endpoint of the `Raw Data Extractor MS`, implemented through the proxy. Serving as the primary orchestrator, the `Raw Data Extractor MS` ensures the correct processing of this call. Initially, it requests the raw data source parameters from the `Raw Data Request Parameter Microservice`. Then, it proceeds to extract raw public data from the specified data source via the provided application interface. Upon completion of the extraction process, the data is dispatched to the `Raw Public Data Microservice` for storage. Upon successful completion of all of these steps, the administrator (caller) is notified. The execution halts whenever a step encounters a failure, and an error response is returned. For the execution of the previous calls, multiple endpoints were necessary, detailed as follows. As previously explained, initiating data extraction requires invoking the "/extract-data" endpoint. The caller must provide a data source ID, which corresponds to the ID stored in the raw data request parameter service. The pseudo code for this endpoint can be found in the Algorithm 3.2. Upon invocation of Algorithm 3.2, the received ID is relayed to the manager when invoking the *executeRawDataExtraction* method. The `manager class` returns a positive or

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

negative response along with the appropriate response code upon receiving the result. Currently, our prototype returns an error code 400 in the case of an exception. In the final product implementation, distinct error codes for potential errors must be incorporated.

Algorithm 3.2: Extract Data endpoint

```
1  request, response
2  resp ← executeRawDataExtraction(request.body.id)
3  if resp = error then
4  |   response.status = 400
5  |   response.send = "error"
6  end
7  else
8  |   response.status = 200
9  |   response.send = resp
10 end
11 return response
```

To ensure that `Raw Data Extractor` effectively retrieves the necessary request parameters for each specific request, it relies on the `Raw Data Request Parameter service`. This crucial interaction is facilitated by the function outlined in Algorithm 3.3. Within this method, a REST call is initiated to the `Raw Data Request Parameter Service`, which contains essential elements such as the service URL, parameter identifiers, call headers, and the request method. Responses are handled differently depending on whether an asynchronous or synchronous implementation is chosen. For our project, we have chosen an asynchronous approach to execute this method. Through this decision, we ensure optimal system performance and a responsive system, which is confirmed by our experiments.

Algorithm 3.3: Get Request Parameters method

```
1  id, serviceUrl
2  method ← GET
3  headers ← Content − Type : application/json
4  result ← fetch([serviceUrl]/get/[id], method, headers)
5  return result
```

As can be seen on line four in the 3.3 algorithm, the endpoint called is a /get/[id] endpoint. This endpoint needs an id number of the database entry required. If an entry with the id is found, the whole database entry is returned.

In Algorithm 3.4, we present the pseudo code illustrating the process of fetching raw public data. Obtaining this data requires both the URL and the request parameters, both of which are obtained from the `Request Parameter Microservice`.

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

Algorithm 3.4: Extract raw data from source

1 $url, requestParameters$
2 $result \leftarrow fetch(url, requestParameters)$
3 return $result$

Once the raw data is successfully retrieved, it needs to be stored. This entails making a call to the `Raw Data microservice`, facilitated by a function outlined in Algorithm 3.5. The URL required for the call is created by this function. The raw data service URL is stored in an environment variable, followed by the path "/add-all/" to access the appropriate endpoint for adding all data, and supplemented with the raw data model, which indicates the database model name and, therefore, the raw data table. Database models are sourced from the request parameter database, along with other parameters. Then, a call to the `Raw Data Microservice` is formulated. This call is then returned to the invoked function, where it is processed accordingly. Depending on the implementation, further actions such as awaiting a response may be necessary.

Algorithm 3.5: Save Raw data method

1 $data, dataModel, serviceUrl$
2 $callUrl \leftarrow [serviceUrl]/add - all/[dataModel]$
3 $method \leftarrow POST$
4 $headers \leftarrow Content - Type : application/json$
5 $body \leftarrow json(data)$
6 $result \leftarrow fetch(callUrl, method, headers, body)$
7 return $result$

Algorithm 3.6: REST endpoint example - ADD

1 $dataModel, requestBody, response$
2 $resp \leftarrow addData(dataModel, requestBody)$
3 **if** $resp = error$ **then**
4 $\quad response.status = 400$
5 $\quad response.send = "error"$
6 **end**
7 **else**
8 $\quad response.status = 200$
9 $\quad response.send = resp$
10 **end**
11 return $response$

The different rest endpoints mentioned in the REST endpoints chapter 3.3.4 are all situated in the communications file. An example of an endpoint is displayed in 3.6. This

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

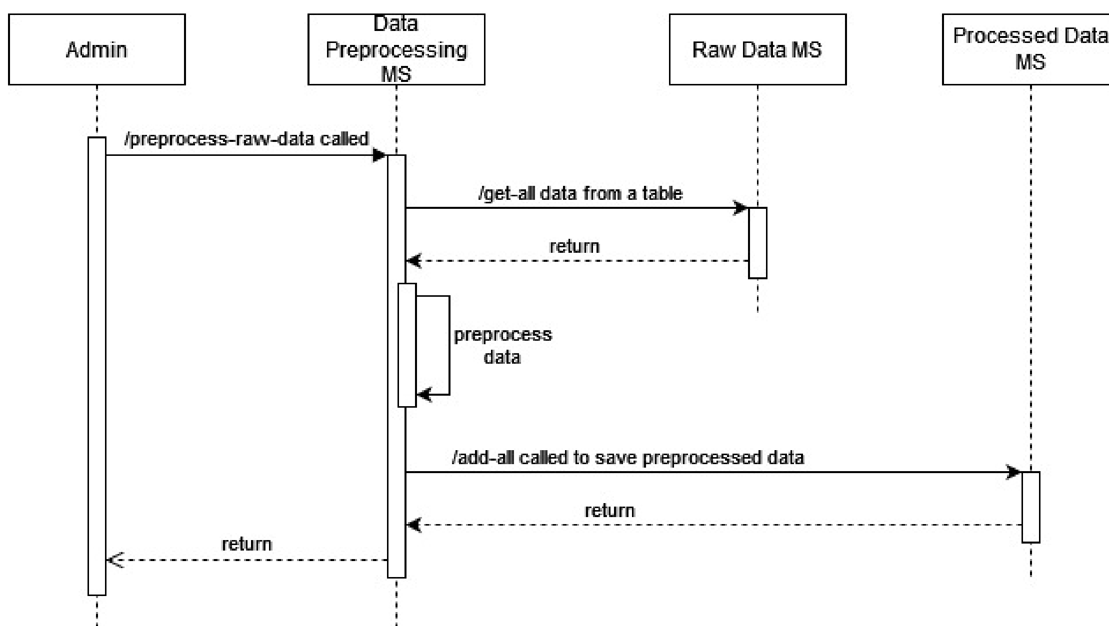


Figure 3.12: Preprocess Raw Public Data Sequence Diagram

endpoint passes the call on to the responsible function inside manager file and handles the response. Since this is a prototype of the final system, a very simple error handling was implemented. This error handling returns status code 400, if an exception occurred and 200 otherwise.

3.3.2 Preprocess Raw Public Data Communication

For raw public data preprocessing multiple steps have to occur. The steps are displayed in Figure 3.12.

After the admin initiates execution by sending a request via the Proxy to the `Data Preprocessing Microservice`, the bulk of execution takes place within this service. As depicted in the figure, the "/preprocess-raw-data" endpoint needs to be invoked. This endpoint expects a JSON body in the request, allowing it to process the data appropriately. You can refer to the data model in code snippet below for the structure of JSON body.

```
{
    "rawDataModel": "ExampleRawTestData",
    "columnsToAdd": [
        {"Source": ""},
        {"City": ""},
        {"Industry": ""}
    ],
```


Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Wien Bibliothek Your knowledge hub

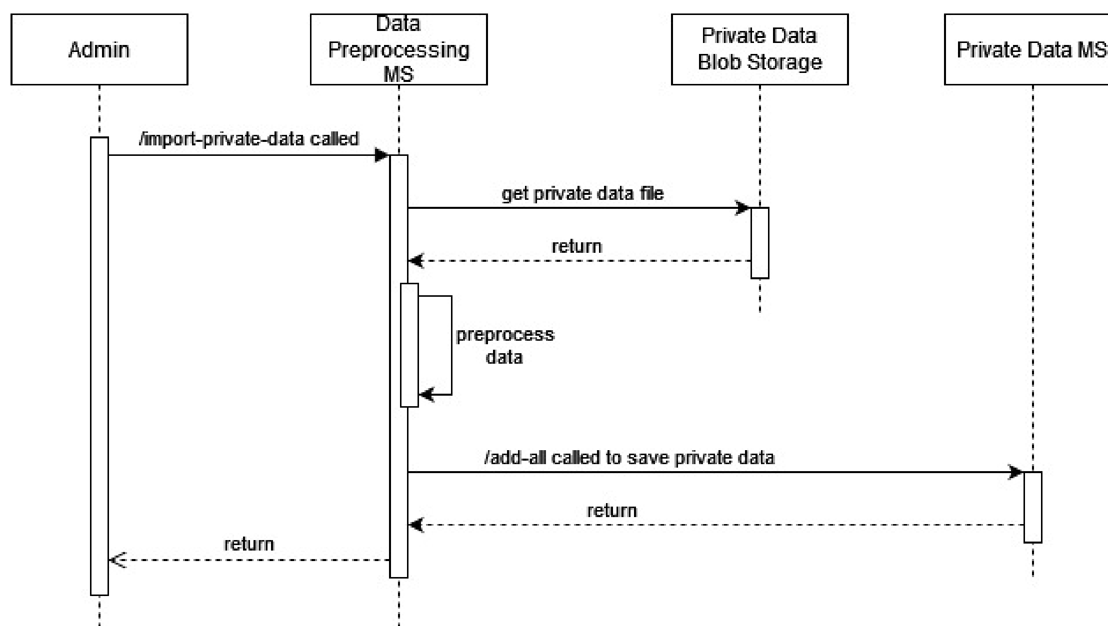


Figure 3.13: Preprocess Raw Private Data Sequence Diagram

```
    "columnsToRename": [
        {"DateTime" : "Year"}
    ],
    "preprocessedDataModel": "ExamplePreprocessedTestData"
}
```

In the above code snap model, the `rawDataModel` is the data model of the source database, being the Raw Data Database. `ColumnsToAdd` and `columnsToRename` are columns which need to be added and renamed respectively. `PreprocessedDataModel` is the database model name in the Preprocessed Data Database. With the help of the `rawDataModel` parameter, the raw public data is received from the `Raw Public Data Microservice`. The endpoint to get the data is "/get-all-data". After the data has arrived in `Data Preprocessing microservice`, it is preprocessed and then sent to the `Processed Data Microservice` for storing it. "/add-all" endpoint from the Processed Data Microservice is called with the data to save and the `preprocessedDataModel` as a parameter to execute the data saving. Finally, the execution results are returned to the caller.

3.3.3 Preprocess Raw Private Data Communication

The execution sequence for preprocessing raw private data is similar to how the raw public data is preprocessed.

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

The figure 3.13 shows the sequence of preprocessing the private data. The harmonization is started by an admin, who calls the "/import-private-data" endpoint from the Data Preprocessing Microservice and passes the data mentioned in the following data model.

```
{
    "privateDataFileName": "full_dataset.xlsx",
    "columnsToAdd": [
        {"Source": "Source Name"},
        {"City": "Vienna"},
        {"Industry": "Electricity, gas, steam..."},
        {"Country": "Austria"}
    ],
    "columnsToRename":[
        {"DateTime" : "Year"}
    ],
    "privateDataModel": "ExamplePrivateData"
}
```

The microservice needs the data to preprocess. This data is received from the Private Data Blob Storage, which returns the requested file. The name of the file is inside the data model and goes by the `privateDataFileName` parameter. Next step in the sequence diagram shown in figure 3.13 is to preprocess the data. For preprocessing, `columnsToAdd` and `columnsToRename` are used. After the preprocessing has been finished, the data is sent to the Private Data Microservice by using the "/add-all" endpoint. Also here, a parameter needs to be passed, so that the microservice knows, where to save the data. For this `privateDataModel` is required. Finally, the execution result is returned to the admin. If any errors occurred, an error status response is returned. If not, a success response is returned.

3.3.4 REST endpoints for Microservices

For a microservices cloud environment, endpoints for the communication are vital. These give the possibility to save, update and request the data, and to start execution processes.

Raw Data Extractor Microservice endpoints The "/extract-data" REST endpoint is featured by `Raw Data Extractor Microservice`. This endpoint initiates the workflow execution, requiring an ID to specify the raw data source to be accessed. Once the entire workflow has been successfully completed with positive responses, a status of 200 is returned along with the number of newly saved database entries. Alternatively, if an error occurs during the workflow, a 400 error code is returned. It is necessary to check the console for error messages as this is a prototype.

Raw Data Request Parameter Microservice endpoints This microservice has the following endpoints:

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

- "/add" - used to add new request parameters. This endpoint receives the url, a name, a description, `requestParameters` and rawDataModel in a JSON format to save them in the database
- "/get/[id]" - this endpoint is used to request parameters for source database APIs with the help of an id. Instead of [id], the integer id of the target database entry has to be entered when the request is made. The id is generated automatically, when a new data record over "/add" endpoint is created.
- "/get-all" - receive all request parameter entries currently existing in the request parameter database
- "/update" - used to update a specific data entry. Since the unique identifier is generated automatically, it has to be included in the request body in addition to the other partially optional parameters mentioned in the database model
- "/delete/[id]" - to delete a specific entry, the "/delete/[id]" endpoint is used. Instead of [id] an integer of the database entry, which has to be deleted, needs to be passed.

Data Preprocessing Microservice endpoints There exists a different set of endpoints for `Data Preprocessing Microservice`. These are used to preprocess raw public data and import private data into the private data database. The implemented endpoints are as follows:

- "/preprocess-raw-data" - this endpoint is used to import raw data from `Raw Data Database` into the `Processed Data Database` while adding additional columns and/or renaming them. Furthermore, empty entries and double entries are considered.
- "/import-private-data" - for preprocessing and importing private data, this endpoint is used. Similarly as for the preprocess raw data endpoint, also this endpoint adds and/or renames columns and/or cleans up the empty and double entries.

Public Raw Data Microservice endpoints Similar to the endpoints in the `Raw Data Request Parameter Microservice`, there are several CRUD endpoints for the `Public Raw Data Microservice`. The `dataModel` request parameter is required for all of these endpoints in order to specify the data model, which is used to identify the table, where the data is considered. The implemented REST endpoints include:

- "/add/[dataModel]" - this endpoint is used to create one new database entry. Since the `dataModel` is passed in the request parameter, the specific required parameters (columns) for the data entry have to be included and may differ from one data model to the next one.
- "/add-all/[dataModel]" - to create multiple data entries in the database, "/add-all/[dataModel]" is used. This endpoint creates new data records from the received request body, which need to include entries matching the dataModel.
- "/get/[dataModel]/[id]" - to request a single entry from the database from a specific table, this endpoint has to be used. An integer id number is required to get the specific entry. This id is automatically generated as the data record

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

is created in the database.
- "/get-all/[dataModel]" - depending on the `dataModel`, all of the entries from a specific table are returned.
- "/update/[dataModel]" - to update a single entry, the "/update/[dataModel]" endpoint has to be used. The data is passed in the body. The body not only requires the data matching the initial parameters, but also an integer id, which is used to identify the entry to update.
- "/delete/[dataModel]/[id]" - this endpoint deletes a single entry from the database. The identification of this data record occurs with the help of `dataModel` and `id` URL parameters. Both of them are required for a successful entry removal.

Processed Data Microservice endpoints The `Processed Data Service` has the same endpoints as `Raw Data Microservice` has. The difference is, that it is handling the preprocessed public data instead of the raw public data.

Private Data Microservice endpoints For `Private Data Microservice` the endpoints are the same as for the `Raw Data Microservice`. The difference is, that Private Data Microservice works with preprocessed private data.

3.3.5 Proxy for Harmonization System

Reverse proxy servers are often used in cloud systems to manage access to microservices and provide centralized access points for users. Using this proxy, incoming requests can be intelligently routed to specific microservices according to their paths. Once a request reaches the proxy, it forwards it to the designated microservice. Following receipt of a request, microservices can interact either through the proxy or directly, depending on the system's implementation guidelines. Users, including clients and administrators, submit their requests specifying the target microservice and its endpoint to the proxy, which then facilitates the forwarding process. Subsequently, the proxy also handles sending the response back to the client, ensuring seamless communication and operation within the system.

3.4 Data Preprocessing

Data preprocessing, or harmonization, is an essential component of the proposed architecture. It simplifies the process of extracting knowledge from a wide range of private and public data sources. Data harmonization entails the amalgamation of data originating from various sources[51]. The process of using these combined data is divided into two steps. The first is the preprocessing of the data, followed by the generation of new knowledge. Since the second step depends on the data at hand and a clear vision of what data needs to be generated, human interaction is needed. It is therefore not possible to reach a general level of automation in this step. As part of the preprocessing step, general data harmonization steps may be used for data preprocessing. Since these steps can be generalized over multiple different data sources and data sets, this preprocessing

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek WIEN Your knowledge hub

can be done automatically and thus implemented in a service. For the architecture at hand, the actions done to the data during preprocessing are concentrated inside the Data Preprocessing Microservice.

3.4.1 Raw Public Data Preprocessing

This microservice consists of multiple classes and methods, which are shown in Figure 3.14 and discussed below.

Figure 3.14 explains the class diagram for the raw public data preprocessing. The methods and classes used for preprocessing the raw private data will be explained as follows:

Communication Layer (number 1) is the same as for the other microservices discussed in previous sections.

Communicator Class (number 2) has a similar task to the communicators explained in other microservices. This is - exchanging requests with other services and functioning as a link between other microservices and the Raw Public Data Preprocessing microservice.
The exact functionalities of the communicator class are:

- The endpoint "/preprocess-raw-data" is one of the endpoints this microservice has. It is used to begin the preprocessing of raw data.
- To execute preprocessing, the raw data must be requested from Raw Public Data microservice. This is done with the help of `GetPublicRawDataJson(...)` method. The data is returned in JSON format.
- After the execution is finished, the raw raw data needs to be stored. This operation is done using the `SavePublicPreprocessedData(...)` method. It calls the `Public Preprocessed Data microservice` to persist the data.

To start the preprocessing of raw public data, the "/preprocess-raw-data" endpoint needs to be invoked.
The parameters for calling the "/preprocess-raw-data" endpoint are as follows:

- rawDataModel - this is the name of the data model and table name used in the Public Raw Data microservice.
- emptyEntriesShouldBeRemoved - a boolean value for determining, whether empty rows in the dataset should be removed
- doubleEntriesShouldBeRemoved - a boolean value for determining, whether duplicate rows in the dataset should be removed
- columnsToAdd - are the columns, which need to be added to the dataset during the preprocessing
- columnsToRename - this is a JSON array consisting of "key:value" pairs, where the "key" is the old column name and "value" is the new column name
- processedDataModel - this parameter is the model and table name used in the Processed Data microservice to identify, in which table the data needs to be persisted

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

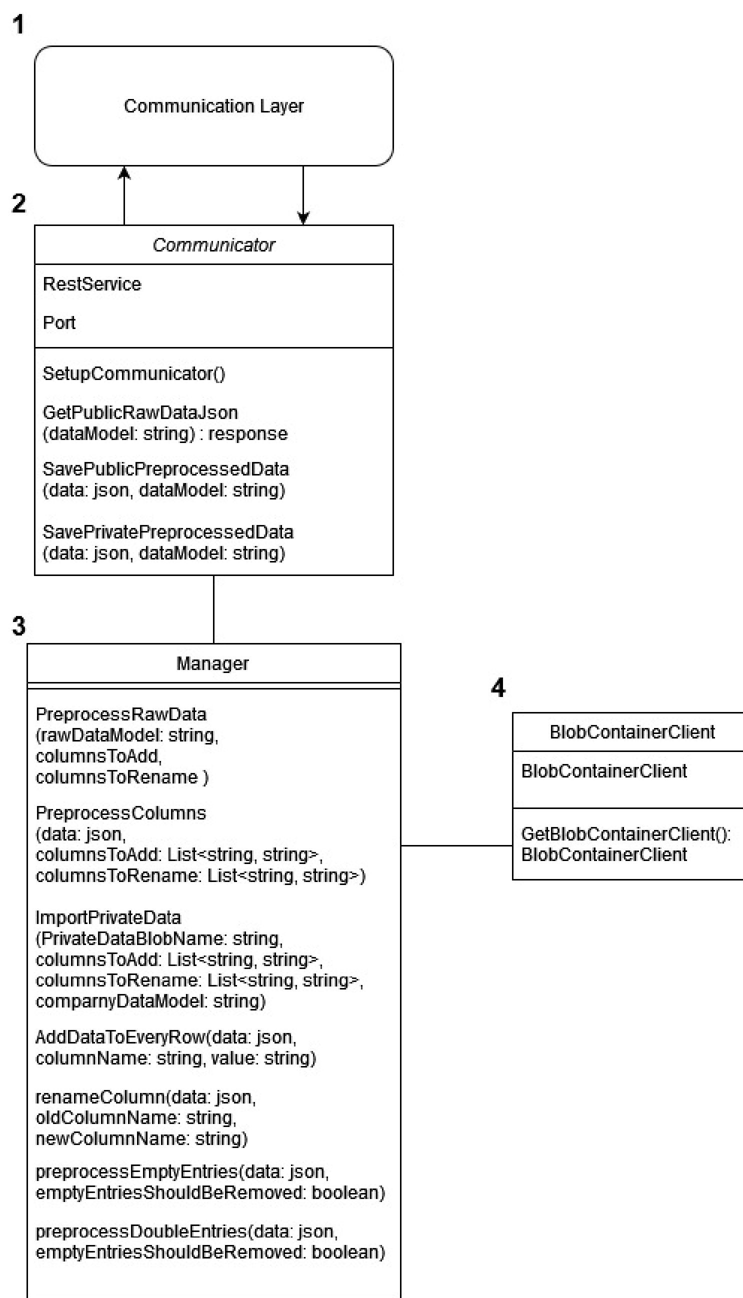


Figure 3.14: Data Preprocessing Microservice Class Diagram

The actual preprocessing occurs inside the `Manager class` (number 3). It is essential to have the relevant dataset available for preprocessing raw public data. This data is acquired through the `GetPublicRawDataJson(...)` method, which invokes the Raw Public Data microservice to retrieve the necessary data in JSON

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

format. This method is invoked by the `Manager class`, which subsequently receives the data. After the execution of this phase, the preprocessed data needs to be stored. This operation is done using the `SavePublicPreprocessedData(...)` method. It calls the `Public Preprocessed Data microservice` to persist the data.

Manager Class The Manager Class (number 3) is central hub for data harmonization. For the harmonization of raw public data the following methods were implemented - PreprocessRawData, RemoveEmptyEntries, RemoveDoubleEntries, Preprocess-Columns, AddNewColumn and RenameColumn.

Algorithm 3.7: Preprocess raw public data

1. $rawDataModel, columnsToAdd, columnsToRename,$ $emptyEntriesShouldBeRemoved, doubleEntriesShouldBeRemoved$
2. $rawDataJson \leftarrow getRawDataJson(rawDataModel)$
3. $rawDataJson \leftarrow$ $removeEmptyEntries(rawDataJson, emptyEntriesShouldBeRemoved)$
4. $rawDataJson \leftarrow$ $removeDoubleEntries(rawDataJson, doubleEntriesShouldBeRemoved)$
5. $rawDataJson \leftarrow$ $preprocessColumns(rawDataJson, columnsToAdd, columnsToRename)$
6. $response \leftarrow savePublicPreprocessedData(rawDataJson)$
7. return $response$

PreprocessRawData method is shown in Algorithm 3.7. This method orchestrates the prepocessing of the raw public data. It is called from the Communicator class and receives the following parameters - `rawDataModel`, `emptyEntriesShouldBeRemoved`, `doubleEntriesShouldBeRemoved`, `columnsToAdd` and `columnsToRename`. These parameters are sent to the `Data Preprocessing Microservice` by the user when calling "/preprocess-raw-data" endpoint. These parameters are explained in detail as follows:

- The `rawDataModel` corresponds to the name of the table, where the raw data is saved inside the Public Raw Data Microservcie. This data needs to be requested from the Raw Public Data microservice. During the preprocessing this is also the data, which is being preprocessed. The `rawDataModel` parameter is a must-have requirement for the HTTP request to the Raw Data Microservice and is used in line two of the algorithm 3.7.
- `ColumnsToAdd` parameter is a JSON array, which consists of the column names and their initial values. These columns need to be added to the raw data during the preprocessing.
- `ColumnsToRename` includes a JSON array. This JSON array includes the columns, which need to be renamed. This occurs in a key: value fashion, where key is the old column name and value is the new column name.
- If in dataset appearing empty entries should be found and removed, then

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

the boolean value for `emptyEntriesShouldBeRemoved` needs to be set to `true`. If it is set to true, then the algorithm for empty entry removal 3.8 is executed.

- `DoubleEntriesShouldBeRemoved` is the parameter responsible for removing any duplicates, that might be in the data set. Set this boolean to true to execute the algorithm 3.9.

Initially, the execution requests the raw public data from the Raw Data microservice. This is described in line two of algorithm 3.7. After the data has been received, harmonization can be started. The first step is to remove empty data rows (line three, algorithm 3.7). Afterwards, the duplicate entry removal can be executed (line four, algorithm 3.7). After the empty and double entries have been removed, the column preprocessing can be started. This occurs during the `PreprocessColumns` execution (line five in algorithm 3.7).
Finally, after the preprocessing has been completed, the harmonized data needs to be saved. To do this, the data is sent to the Processed Data microservice. This is done with the help of `savePublicPreprocessedData(...)` method on line six, seen in algorithm 3.7. This methods calls the Communicator, which further creates the HTTP request and sends it to Processed data microservice. After the response from Processed Data microservice has been received, it is returned back to the user of the system, who initiated the harmonization process. This is required, so that the caller sees, whether the execution was successful or not. Furthermore any errors, which might have occurred during the execution, can be viewed.

Algorithm 3.8: Remove Empty Entries

```
1 data, emptyEntriesShouldBeRemoved
2 result = []
3 if emptyEntriesShouldBeRemoved == True
4     foreach row in data
5         foreach cell in row
6             if cell != null
7                 result.add(row)
8                 break
9 return result
```

A simple implementation to remove empty rows is displayed in algorithm 3.8. The parameters for this method are the following:

- The `data` variable represents the JSON data object, which needs to be adapted
- `emptyEntriesShouldBeRemoved` is the variable, which determines, whether the dataset should be searched for empty entries and whether these empty entries should be removed

After it has been determined, whether empty entries should be removed or not on line three in algorithm 3.8, the received data object is being iterated over. During

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

this iteration on lines four and five each of the entry cells are reviewed. If the reviewed cell is not empty (line six, algorithm 3.8), then the row is added to the resulting data array on line seven in algorithm 3.8, and the execution for this row is stopped with a break statement (line eight, algorithm 3.8). After all of the entries have been processed, the resulting data is returned on line nine. It is important to remove empty entries, because failing to do so can result in multiple negative effects. These empty rows carry no extra value, but increase the memory occupied. Furthermore, during processing of the dataset, the empty entries use up CPU time, since they also need to be considered. In this way, empty entries increase the cost of all processes they are involved in. Thus the removal of empty entries is necessary, so that the memory occupancy and CPU time requirements are reduced.

Algorithm 3.9: Remove Double Entries

1 $data, doubleEntriesShouldBeRemoved$
2 $result = []$
3 if $doubleEntriesShouldBeRemoved == True$
4 $\quad createdSet \leftarrow$ new $Set(data)$
5 $\quad result \leftarrow createdSet.ToJson()$
6 return $result$

The removal of duplicate entries can have similar benefits as when removing empty entries, mainly the reduction of processing time needed and the reduction of memory occupancy. The algorithm 3.9 displays the `RemoveDoubleEntries` method. To call this method, the following parameters are required:

- `data` represents the raw dataset at hand, which needs to be processed
- `doubleEntriesShouldBeRemoved` variable is needed, so that it can be determined, whether double entries should be removed or not

The first step during the execution is to determine, whether double entries should be removed. This occurs on line three in algorithm 3.9. If duplicate rows are to be removed, then there are multiple options how to do it. One of them is to use a Set object. A set is a container, which contains every element only once. Thus, if there are two elements with all matching values, the result set would contain this element only once. The creation of the set is displayed on line four in algorithm 3.9. The reversion of set to JSON happens on line five in algorithm 3.9. After the conversion from set back to JSON has been finished, the results are returned on line six in algorithm 3.9.
PreprocessColumns method as shown in Algorithm 3.10 is the next step during the preprocessing. This method receives the following parameters:

- `data` - raw data, which is to be preprocessed
- `columnsToAdd`
- `columnsToRename`

This `PreprocessColumns` method orchestrates the preprocessing of the columns, which perform following two operations.

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

Algorithm 3.10: Preprocess Columns

```
1  data, columnsToAdd, columnsToRename
2  if columnsToAdd != null
3      foreach column in columnsToAdd
4          columnName ← column.key
5          data ← AddNewColumn(data, columnName, column[columnName])
6  if columnsToRename != null
7      foreach column in columnsToRename
8          columnName ← column.key
9          data ← RenameColumn(data, columnName, column[columnName])
10 return data
```

- adding columns - is the process of adding new, not yet existing columns to the data set. These could be for example year, country or industry. As mentioned before, the variable columnsToAdd is a JSON array, which also can be empty or null. Inside the PreprocessColumns algorithm 3.10 this adding columns process is displayed in lines two to five. On line two it is made sure, that the columnsToAdd value is not null and thus new columns must be added. On line three in algorithm 3.10, the for loop for adding all received columns is initiated. After the for loop has been initiated, `AddNewColumn` 3.11 algorithm is called on line five (see algorithm 3.10). For this algorithm the column the columnName is extracted from the JSON array.
- renaming columns - is a very similar process to the process of adding new columns. First, it is checked, whether columnsToRename JSON array is null on line six in algorithm 3.10. Afterwards, each of the columns is renamed in a similar fashion as for column renaming on lines seven to nine (algorithm 3.10).

After the data has been adapted, it is returned on line ten in algorithm 3.10.

Algorithm 3.11: Add New Column

```
1  data, columnName, value
2  foreach dataRow in data
3      dataRow[columnName] ← value
4  return data
```

Algorithm 3.11 displays the `AddNewColumn` method, which is called during the harmonization process by the `PreprocessColumns` method explained before. This method requires three parameters - `data`, `columnName` and `value`. `columnName` is the name of the column, which needs to be added. `data` holds the data, to which the column needs to be added. Depending on the data structure in `data` different procedures for adding the new column can be used. One of them is to iterate over every entry and add the new key:value pair. Key being the name of

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

the column and value the value to be inserted. Adding new columns during the harmonization process is important, so that the data can be analyzed based on these added columns at a later point in time. For example, adding country, city, year or industry to different data sets can help with the analyses of these data sets.

Algorithm 3.12: Rename Column

1 $data, newColumnName, oldColumnName$
2 foreach $dataRow$ in $data$
3   $dataRow[newColumnName] \leftarrow dataRow[oldColumnName]$
4   $dataRow[oldColumnName] \leftarrow null$
5 return $data$

The algorithm 3.12 shows the `RenameColumn` method, which is used to give an already existing column a new name. This can be useful to make sure, that for all of the data sets, columns containing the same types of values, have the same column name. This is important, so that the further data analysis can be conducted in an easier manner.
So that the renaming process can be done, three variables are necesssary:

- `data` - consisting of the data, where the column is to be renamed
- `oldColumnName` - the name of the column, which has to be renamed
- `newColumnName` - the name of the column after it has been renamed

In the algorithm 3.12, a loop iterates over every row in the received data (line two) and assigns the value from the old column to the new column (line three in algorithm 3.12). After this has been done, on line four (algorithm 3.12) the data from the old column is removed. As a last step the result is returned on line five.

To further improve the readability and structure of the code, AddColumn and RenameColumn methods can be moved to another class, which is solely used for hosting preprocessing methods. For the simplicity of the thesis it was chosen to include them inside the Manager class.

3.4.2 Raw Private Data Preprocessing

For the preprocessing of Raw Private Data the same figure 3.14 as in the previous subsection is used. Raw Private Data Preprocessing concerns the harmonization of private data. This data can originate eg. from companies or universities and it is not publicly available and should not be made publicly available. Thus, here importing data from data files, which are persisted inside a blob storage, was concerned.

Following are the explanations for the classes and methods used for preprocessing of private data:

Communication Layer This is the same as for the other previously mentioned microservices.

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek WIEN Your knowledge hub

Communicator Class To begin the private data harmonization execution, "/import-company-data" must be called. For this endpoint similar parameters are required as for the previously explained "/preprocess-raw-data" endpoint. These are:

- dataFileName - this is the name of the file, where the raw data is saved
- emptyEntriesShouldBeRemoved - a boolean value for determining, whether empty rows in the dataset should be removed
- doubleEntriesShouldBeRemoved - a boolean value for determining, whether duplicate rows in the dataset should be removed
- columnsToAdd - are the columns, which need to be added to the dataset during the preprocessing
- columnsToRename - this is a JSON array consisting of `key:value` pairs, where the `key` is the old column name and `value` is the new column name
- processedDataModel - this parameter is the model/table name used in the Private Processed Data microservice to identify, in which table the data needs to be persisted

After the preprocessing of private data has been finalized, it needs to be persisted. To do so, the data has to be sent to Private Processed Data Microservice. This occurs in the `SavePrivateDataPreprocessedData(...)` method, after the Manager class has finished the preprocessing.

Manager Class To harmonize private data, the following methods are utilized: `ImportPrivateData`, `PreprocessColumns`, `RemoveEmptyEntries`, `RemoveDoubleEntries`, `AddDataToEveryRow` and `RenameColumns`. Since `PreprocessColumns`, `AddDataToEveryRow`, `RemoveEmptyEntries`, `RemoveDoubleEntries` and `RenameColumns` methods have already been introduced earlier, they won't be talked about here one more time.

Algorithm 3.13: Import private data

1. $dataFileName, columnsToAdd, columnsToRename,$ $emptyEntriesShouldBeRemoved, doubleEntriesShouldBeRemoved$
2. $privateDataJson \leftarrow downloadAndOpenFile(dataFileName)$
3. $privateDataJson \leftarrow$ $removeEmptyEntries(privateDataJson, emptyEntriesShouldBeRemoved)$
4. $privateDataJson \leftarrow$ $removeDoubleEntries(privateDataJson, doubleEntriesShouldBeRemoved)$
5. $privateDataJson \leftarrow$ $preprocessColumns(privateDataJson, columnsToAdd, columnsToRename)$
6. $response \leftarrow savePrivatePreprocessedData(privateDataJson)$
7. return $response$

ImportPrivateData's method, displayed in algorithm 3.13, is used only during the harmonization of private data. This method has the following parameters:

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

- *dataFileName* - is the name of the file, from where the data needs to be imported
- *emptyEntriesShouldBeRemoved* - a boolean value for determining, whether empty rows in the dataset should be removed
- *doubleEntriesShouldBeRemoved* - a boolean value for determining, whether duplicate rows in the dataset should be removed
- *columnsToAdd* - is a JSON array consisting of columns, which need to be added
- *columnsToRename* - is a JSON array, which includes the columns to rename

On line two of the algorithm 3.13, the downloading and opening of the data file occurs. For this, the Blob Container Client, explained further down, is utilized. After the file has been downloaded, it is opened and loaded into the `privateDataJson` variable.

The subsequential data preprocessing steps, data saving and returning the results are the same as for previously explained `PreprocessRawPublicData` and thus won't be discussed in more detail here.

BlobContainerClient is utilized to establish communication with blob storage, where private data files are stored. This involves configuring communication strings, authentication, and any other requisite elements to enable the `Manager class` to access the data files.

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.

CHAPTER 4

Results

This chapter provides an analysis of our proposed harmonization system. Initially, we provide setup including technology stack used and system deployment. Afterwards data analysis process and effect on memory and computational times are analyzed and compared. We also discuss various datasets used to analyze the system and their data models.

4.1 Experimental setup

In this section, we provide a detailed discussion regarding the technology used, data models (including both raw and processed data)and deployment strategies.

4.1.1 Technology Stack

This section highlights the essential role of tools and technologies in ensuring smooth microservices functioning. In addition, it provides succinct descriptions of the specific tools used in this study. The technology stack used in this work is as follows.

NodeJS was used for programming each of the microservices in the system.
Makefiles were written to be able to easily start the services locally using Make.
PostgreSQL was the SQL database used as the main database technology.
Azure Storage Account was used as a place to upload the company data raw files. Blob storage was the type of storage account utilized.
Azure is the general cloud service environment, where the whole cloud environment was setup. On Azure the following services were used:

- Azure DevOps
- Container Registry
- Kubernetes service
- Azure Database for PostgreSQL

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

- Storage Account
- Key Vault

Azure DevOps is a version control tool from Microsoft. It was not only used for managing code, but also to create a pipeline for generating the docker images and deploying them on Azure Kubernetes.

Azure Container Registry is the storage, where docker images were uploaded after creation and later down the line downloaded for deployment.

Docker was used to package and deploy the different microservices.

Kubernetes on Azure is the technology used to host the docker containers for the different microservices.

Nginx is a reverse proxy technology, which was applied to access the different services in the kubernetes environment.

Power BI is the tool used for analyzing the different collected and harmonized data.

4.1.2 System deployment

This system had to be deployed to a cluster in order to analyze its current implementation. In this experiment, we used Azure Cloud Services. The following tools were used in this environment:

- *Azure Blob Storage* - for storing data files containing private data, which is imported into the system
- *PostgreSQL* - instances of PostgreSQL were used for the different databases required by the microservices
- *Azure DevOps* and *Azure DevOps Pipelines* - Azure Devops used as a version control system (VCS) and the pipelines used to deploy the implemented architecture with the help of docker containers
- *Azure Container Registry (ACR)* is necessary for storing and providing the docker images for the implemented microservices. These images are pushed to the registry by azure pipelines.
- *Azure Kubernetes Service (AKS)* - used to host the microservices, as they are deployed

The deployment of the system is started by starting the specific *Azure Devops Pipeline*, which creates the *docker* images, starts the kubernetes environmentand orders the deployment of the saved docker images.

Azure DevOps Pipeline

A comprehensive description of the entire process of building, publishingand deploying is presented in this section. It is imperative to initiate the build process for each service before deploying a new version of it or the entire system. The following code listing illustrates the pivotal steps involved in the build process:

```
- stage: BuildRawDataService
```


Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

```
  displayName: Build stage Raw Data Service
  dependsOn:
  jobs:
  - job: Build
    steps:
    - task: Docker@2
      inputs:
        command: buildAndPush
        repository: $(imageRepositoryRawDataService)
        dockerfile: $(dockerfilePathRawDataService)
        containerRegistry: $(dockerRegistryServiceConnection)
```

The above listing illustrates the procedural steps involved in building and pushing a Docker image of the raw data service. Of the utmost importance within this listing are the Docker@2 task, the `buildAndPush` commandand its associated inputs including the repository, Dockerfileand container registry.

After every microservice is built, the publish stage generates Kubernetes manifest files to facilitate further deployment stages. The following listing illustrates the Kubernetes files publishing:

```
- stage: Publish
  displayName: Publish stage
  dependsOn:
    - BuildProcessedDataService
    - BuildRawDataExtractorService
    - BuildRawDataRequestParameterService
    - BuildRawDataService
    - BuildRawDataPreprocessService
    - BuildPrivateDataService
  jobs:
  - job: Publish
    steps:
    - publish: manifests
      artifact: manifests
```

The lines in the listing below initiate the deployment of services, focusing on the Raw Data Service deployment within the listing. It is essential that the Azure Kubernetes Service has access to the Docker images before the service is executed, which is achieved by creating the image pull secret as the first step. A second task involves retrieving a docker image and creating a cluster instance using the docker container created from a pulled image. In these there are several parameters, which are summarized below.

- `Manifests` parameter is where the in the publish stage pushed Kubernetes manifest file is selected.

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

- This parameterizes the previously created docker image pull secret as `imagePullSecret`, which is helpful for accessing the docker image repository.
- Finally, the `containers` parameter corresponds to the location of the referenced docker image. From this location, the Raw Data Service image with the 'latest' tag is pulled.

```
- stage: DeployRawDataService
  displayName: Deploy stage Raw Data Service
  dependsOn:
    - Publish
  jobs:
  - deployment: Deploy
    strategy:
      runOnce:
        deploy:
          steps:
          - task: KubernetesManifest@0
            inputs:
              action: createSecret
              secretName: $(imagePullSecret)
              dockerRegistryEndpoint: $(dockerRegistry-
              ServiceConnection)
          - task: KubernetesManifest@0
            inputs:
              action: deploy
              manifests: |
                $(Pipeline.Workspace)/manifests/
                rawDataServiceDeployment.yaml
              imagePullSecrets: |
                $(imagePullSecret)
              containers: |
                $(containerRegistry)/
                $(imageRepositoryRawDataService):latest
```

Kubernetes files

There are several Kubernetes configuration files required in the Kubernetes cluster for each and individual services. For example, the configuration setup for the Kubernetes manifest file is as follows:

```
apiVersion: apps/v1
kind: Deployment
metadata:
```


Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

```
  name: raw-data-service-deployment
spec:
  selector:
    matchLabels:
      app: raw-data-service-deployment
  template:
    spec:
      containers:
      - name: raw-data-service
        image: raw-data-service:latest
        imagePullPolicy: Always
        ports:
        - containerPort: 8001
      imagePullSecrets:
        - name: sustainista-acr-secret
```

The above manifest file orchestrates the deployment of the Raw Data Service within the Kubernetes cluster. The manifest file type specification, selectors, metadataand, most importantly, the container configuration are notable. This Kubernetes deployment file references a container from the Azure Container Registry (ACR). Specifically, the image pull policy is set to "Always" in order to ensure continuous deployment of the latest version. Kubernetes must also identify the port on which the system within the container operates by specifying the container port parameter (8001 in this case). Further, the image pull secret is specified. It facilitates authentication to the Azure Container Registry, enabling retrieval of the most recent docker image for the running system [52].

The `Service` deployment type enables communication between the microservice running in the Kubernetes cluster and other services. An example of its configuration is shown in the following code listing, specifying its kind, metadataand deployment specifications. Additionally, these specifications include the port and selector used to determine the deployment target of the service [53].

```
apiVersion: v1
kind: Service
metadata:
    name: raw-data-service-deployment
spec:
    ports:
    - port: 8001
    selector:
        app: raw-data-service-deployment
```

In addition to microservice deployments, an ingress controller is essential for facilitating external access to the various services within the network. The following code

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek Your knowledge hub WIEN

excerpt shows a partial implementation of an ingress controller, including rerouting rules specifically for raw data.

```
apiVersion: networking.k8s.io/v1
kind: Ingress
metadata:
  name: nginx-ingress
  annotations:
    nginx.ingress.kubernetes.io/rewrite-target: /$2
spec:
  ingressClassName: nginx
  rules:
  - http:
      paths:
      - path: /raw-data-service(/|$)(.*)
        pathType: Prefix
        backend:
          service:
            name: raw-data-service-deployment
            port:
              number: 8001
```

The ingress configuration file contains annotations, the name of the ingress classand HTTP paths for incoming requests. Notably, annotations specify a rewrite target pattern, which is specified as "$2". In this pattern, the first parameter is removed from the URL, ensuring that subsequent requests are directed appropriately. The first URL parameter determines which service the request is sent to, because it specifies which resource to use. The URL path element must begin with "/raw-data-service", as indicated within the `path` parameter, to route a request to the Raw Data Service. The application will direct the request to the port indicated in the `paths` section of the manifest file if a matching path is found. Moreover, to specify the ingress controller to be utilized, the `ingressClassName` parameter must be defined in the configuration [54].

4.1.3 Data models

This section explains the different data models utilized in the proof of concept implementation. The datasets used in these experiments are collected from industry partner and they belong to different types of sustainability data or fields, which industry partner uses to analyze sustainability data. This subsection provides `Raw Public Data` data models, `Preprocessed Public Data` data models and `Preprocessed Private Data` data models.

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

Table 4.1: Trading Economics data model

AAAABBB	AAAABBBB	
id	Int	@id @default(autoincrement())
Country	String	
Category	String	
DateTime	String	
Value	Decimal	
Frequency	String	
HistoricalDataSymbol	String	
LastUpdate	String	
		@@unique([Country, Category, DateTime])

Table 4.2: Eurostat data model

AAAABBB	AAAABBBB	
id	Int	@id @default(autoincrement())
Value	Decimal	
Category	String	
Country	String	
DateTime	String	
Indicator	String	
Unit	String	
		@@unique([Country, Category, DateTime])

Raw Public Data Data Models

The raw data received from the various public data sources is stored in the `Raw Public Data Database`. Each source has its own data table, or multiple tables, based on the structure of the data. The catalog for the Trading Economics data model (Table 4.1) and Eurostat data model (Table 4.2).

Preprocessed Public Data Data Models

The `Preprocessed Public Data Database` serves as a repository for data preprocessed by the `Data Preprocessing Microservice`. The raw data is derived from the `Raw Data Database`, so the tables within the preprocessed data database are directly related to those in the public raw data database. There are also additional columns in the preprocessed data tables which may remain empty, including attributes such as company, industry, city, countryand date. Columns that already exist are not duplicated. Additionally, existing columns may be renamed to align with standardized database models, ensuring that datasets are harmonized. There is a renaming process to match attributes such as the company, industry, city, countryand date of issue. We then present the extended data models sourced from Trading Economics and Eurostat to provide supplementary information.

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

Table 4.3: Trading Economics preprocessed data model

AAAABBB	AAAABBBB	
id	Int	@id @default(autoincrement())
Country	String	
Category	String	
Year	String	
Value	Decimal	
Frequency	String	
HistoricalDataSymbol	String	
LastUpdate	String	
Company	String	
Industry	String	
City	String	
		@@unique([Country, Category, DateTime])

Table 4.4: Eurostat preprocessed data model

AAAABBB	AAAABBBB	
id	Int	@id @default(autoincrement())
Value	Decimal	
Country	String	
Indicator	String	
Company	String	
City	String	
Industry	String	
Year	String	
Unit	String	
		@@unique([Country, Category, DateTime])

Preprocessed Private Data Data Model

The `Preprocessed Private Data Database` functions as a repository for data acquired from companies and processed by the `Data Preprocessing Service`. In this database, each private data source is represented in a table that mirrors the structure of the preprocessed public data. It also includes the same columns as preprocessed public data, such as company, industry, city, countryand date (of issue), if not already present. Typically, the added columns are populated with relevant data given the well-defined nature of this data source. Table 4.5 illustrates the imported data model for private data received from the industry partner.

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek Your knowledge hub WIEN

Table 4.5: Industry partner private data model

AAAABBB	AAAABBBB	
id	Int	@id @default(autoincrement())
Company	String	
Country	String	
Year	Int	
Industry	String	
City	String	
Description	String?	
Bill	String	
Material	String?	
Product group	String	
Order amount	Decimal	
Order unit	String	
Order net value	Decimal	

4.2 Effect on Memory and Computations

In this section, we perform data analysis and evaluate the effect on memory and processors. We evaluate the effects of harmonization, duplicate eliminationand partial entry on memory and computation for the following three datasets.

- Eurostat data set
- Trading Economics CO2 data set
- Private data data set

Harmonized data is analyzed using PowerBI to estimate CO2 emissions from the private dataset.

During the design, implementationand deployment phases of the proposed harmonization architecture, there were numerous insights and advantages gained, demonstrating that the approach is effective in terms of computational time and memory usage. The following plots (from Figure 4.1 to Figure 4.11) illustrate elapsed time (a proxy for performance) and memory consumption for various datasets comprising different entry counts (50, 100, 120 or 150). Additionally, a comparison is presented between scenarios with duplicate and empty entry removal enabled (ON) versus disabled (OFF).

4.2.1 Effect on Memory

The analysis of the memory consumption was conducted in the following fashion:

- First, the data was prepared for inserting into the raw database. For this, the amount of entries in the dataset were reduced to 50, 100 and 120 or 150. Each of these amounts of entries were saved in a separate CSV file.

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

- Second, the prepared CSV file with the right amount of entries for the required test run was inserted into the Raw Data Database. Since the process of inserting data into the raw data database was not the subject under investigation, the data could be imported with the help of SQL statements.
- Third, the data harmonization was started, for which a data preprocessing HTTP request was sent to the Data Preprocessing Microservice. The parameters for this request were set according to the test case at hand - these are the variables mentioned before.
- Fourth, after the data harmonization was finished, the memory consumption inside the data table was reviewed and saved.
- Finally, the whole process was repeated for all the other variable combinations and the results recorded.

Figures 4.1 to 4.3 shows memory occupation for the different data sets, with multiple parameters. Some of the parameters are summarized below.

- On the x-axis: number of data entries. Tests with three different amount of entries were conducted - with 50, 100 and 120 or 150.
- On the y-axis: memory occupation. For the runs differing memory consumption was measured. The given values are displayed in bytes.
- Each plot in Figure 4.1 to Figure 4.3 contains three metrics such as - the blue line (Memory) shows the initial memory consumption before applied the proposed harmonization. The red line (ON) displays the memory occupied after the harmonization while enabling duplicate and empty entries removed. The brown line (OFF) shows the memory consumption with the duplicate and empty entries removed in parallel to harmonization.
- We also estimate the variations, as we rename columns of the existing data base during harmonization process. We considered these variations between 1, 3and 5. In each plot in Figure 4.1 to Figure 4.3, the sub figures (a), (d)and (g) considered renaming a single column, the sub figures (b), (e)and (h) considered renaming three columns, whereas the sub figures (c), (f)and (i) considered renaming five columns.
- Along with the renaming, we also considered adding new columns during harmonization. We considered adding new column variations between 1, 3 and 5, similar to renaming. In each plot in Figure 4.1 to Figure 4.3, the sub figures (a), (b)and (c) considered adding a single column, the sub figures (d), (e)and (f) considered adding three new columns, whereas the sub figures (g), (h)and (i) considered adding five new columns.
- In summary, Figure 4.1(a) is the results of adding one new column and renaming a column. Similarly, Figure 4.1(b) shows the addition one new column with the renaming of three columns. Figure 4.1(c) shows the addition one new column while the renaming of five existing columnsand so on.

Figure 4.1 shows the resultant memory occupancy for *Eurostat* data set. The plots reveal distinct observations. Notably, both the OFF and ON plots consistently appear above the Memory plot, due to the addition of a fixed number of columns to the *Eurostat* dataset.

TU Bibliothek Your knowledge hub WIEN
Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.

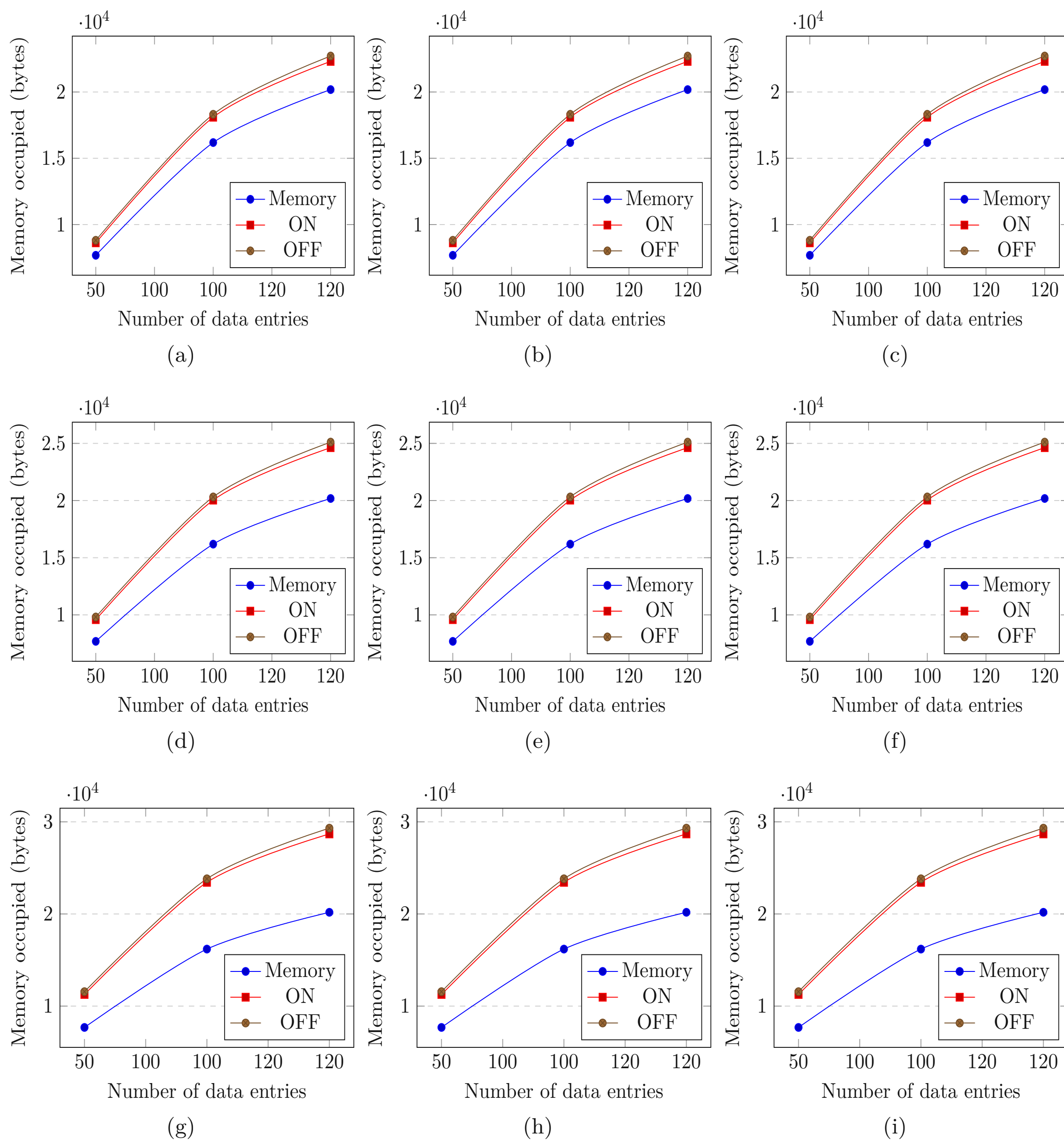


Figure 4.1: Memory occupied for Eurostat data set

Specifically, one, three, or five columns to the dataset. These plots clearly demonstrate the correlation between increased memory usage and the inclusion of additional columns. Nevertheless, due to harmonization and sufficient preprocessing, these minor increases do not affect computation. Further details regarding the percentage increase in memory consumption for the ON plot are provided below:

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

- Increasing the amount of columns by one increased the memory consumption by 10 to 12 %. The higher the amount of entries were, the lower the amount of increased memory consumption was.
- Increasing the amount of columns by three increased the memory occupation by 22 to 24 %. Similarly, as for adding one column, the higher the number of entries were, the lower the increase in percent, vice versa.
- Finally, increasing the amount of columns by five had a similar result as for the other two - the increase in memory was between 42 and 44 %.

The OFF plots exhibited similar trends, with memory consumption ranging from 12 to 14% for 50 entries, 24 to 27% for 100 entriesand 45 to 50% for 120 entries. One to three percent of the difference in memory usage between ON and OFF plots can be attributed to the difference in plots. With the addition of more columns, this discrepancy increases, since the memory requirement for double and empty entries rises at the same rate. Interestingly, the renaming of columns had no discernible effect on the resulting file size. Even as the number of renamed columns increased within a given set of diagrams (e.g., (a), (b)and (c)), memory consumption remained constant. Moreover, the analysis revealed a near-linear relationship between processed entries and memory usage. Doubling the entries processed led to an approximate 2.1-fold increase in required memory, indicating a nearly proportional relationship.

For the *CO2 emissions* dataset, the memory occupation analysis in different amount of entries with varying the number of columns add and/or rename are plotted in Figure 4.2. Here, we also considered removing empty and duplicate entries (ON and OFF plots).

From Figure 4.2, our observations while considering duplicate and empty entries removal effect in terms of percentages of memory consumption increase/decreased is dissed below:

- Increasing the amount of columns by one increased the memory consumption by 7 to 11 %. The higher the amount of entries were, the higher the amount of increased memory consumption was. Opposed to the previous data set.
- Increasing the amount of columns by three increased the memory occupation by 21 to 26 %. Similarly, as for adding one column, the higher the number of entries were, the higher the increase in percent.
- Finally, increasing the amount of columns by five had a similar result as for the other two - the increase in memory was between 56 and 57 %.

Similar observations were noticed when we did not consider removing duplicates or empty entries from the *CO2 emissions* dataset. The range of memory consumption increase for 50 entries was around 16 %, for 100 entries around 31 % and for 150 entries between 58 and 59 %. However, these values are higher when compared with removing duplicates and empty entries from the datasets, which leads to a dramatic reduction in memory usage.

Memory usage varied from three to eight percent between ON and OFF plots, increasing slightly as more columns were added. There was a noticeable reduction in this discrepancy with more entries. For instance, the difference was eight percent for 50 entries but reduced

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Wien Bibliothek Your knowledge hub

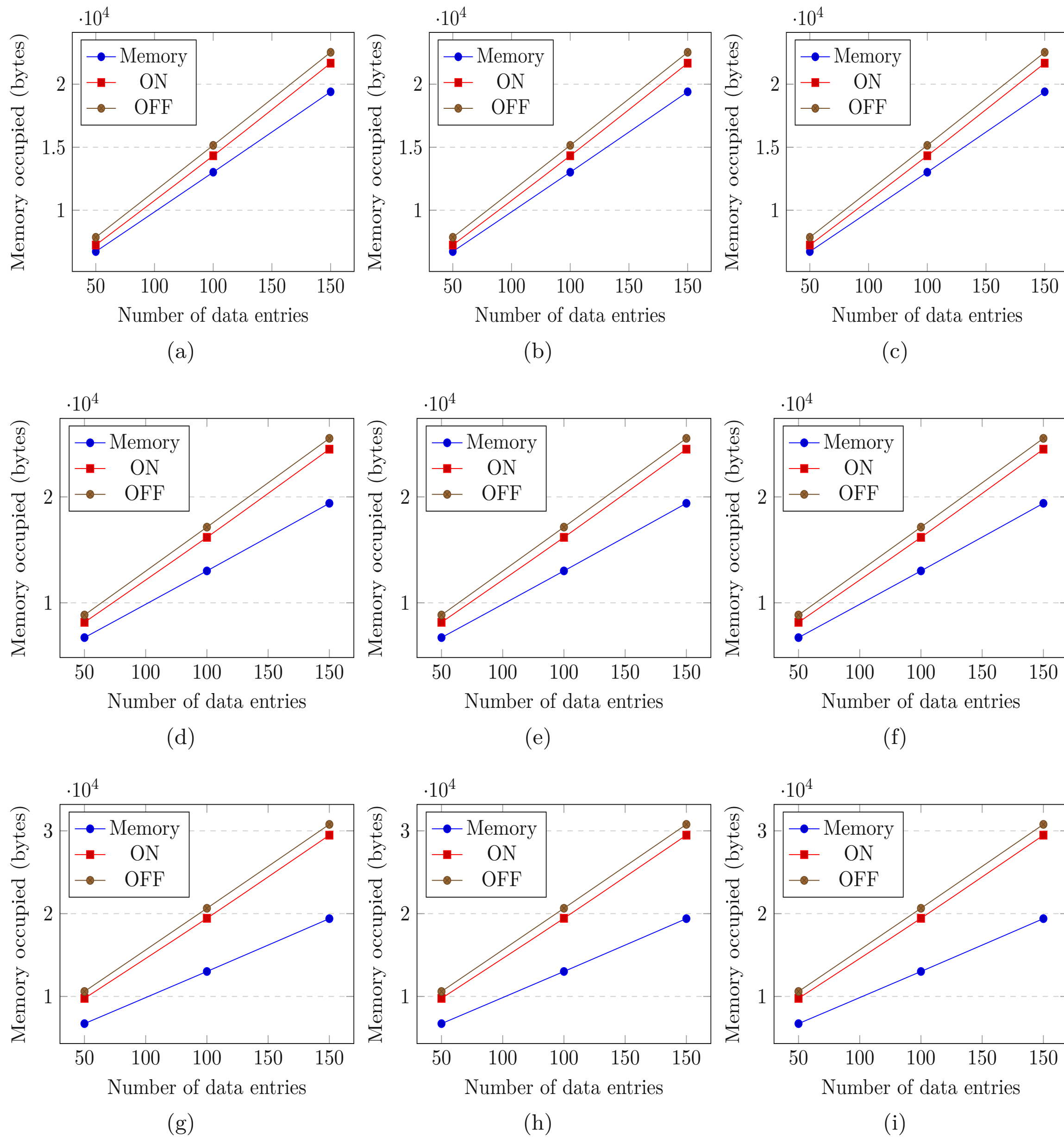


Figure 4.2: Memory occupied - CO2 emissions

to three to four percent for 150 entries. This can be attributed to the fact that while memory consumption increased with more entries, the absolute difference between ON and OFF plots remained relatively stable or changed at a slower rate. Furthermore, renaming columns had no significant impact on the resulting file size, with memory consumption remaining constant despite an increase in the number of renamed columns. Finally, there was a near-linear relationship between processed entries and memory usage.

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

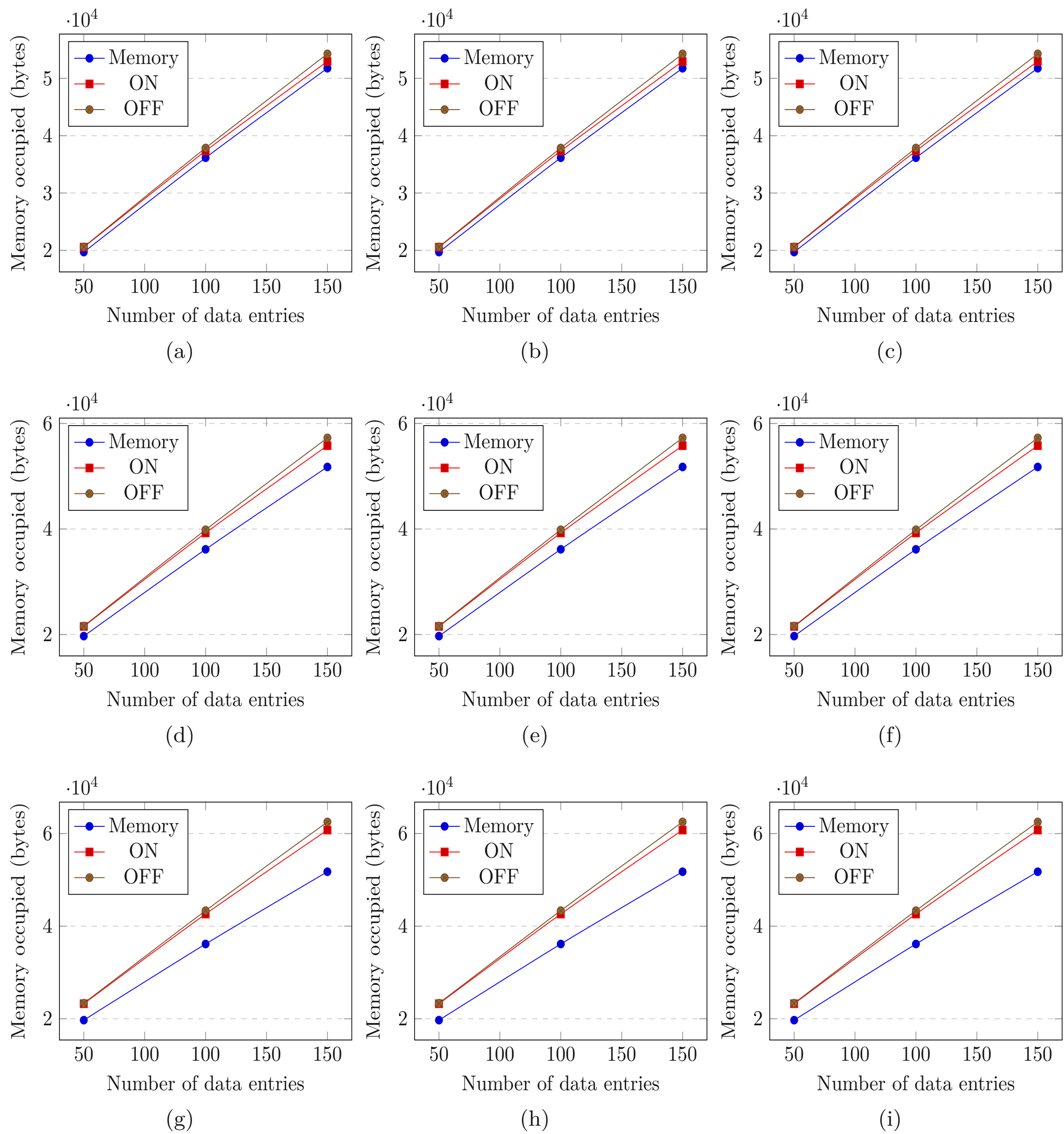


Figure 4.3: Memory occupied - private data dataset

Doubling the entries processed led to an approximate 1.94-fold increase in required memory, indicating a nearly proportional relationship.

Figure 4.3 shows the memory occupation analysis for the private datasets, while varying the number of entries, columns, along with removing and not-eliminating duplicates and empty entries. The observations were made on percentages of memory consumption vari-

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek WIEN Your knowledge hub

ations while eliminating (ON) duplicates and empty entries are plotted in Figure 4.3and a summary of discussions are below.

- Increasing the amount of columns by one increased the memory consumption by two to four percent. The higher the amount of entries were, the lower the amount of increased memory consumption was. This was the same for the first dataset analyzed.
- Increasing the amount of columns by three increased the memory occupation by seven to nine percent. Similarly, as for adding one column, the higher the number of entries were, the lower the increase in percent.
- Finally, increasing the amount of columns by five had a similar result as for the other two - the increase in memory was between 17 and 18 %.

Memory consumption for (OFF) plots slightly increased by approximately four percent for 50 entries, between nine and eleven percent for 100 entriesand between 18 and 21% for 150 entries, as shown in figure 4.3. Memory usage differed between ON and OFF plots by almost zero to three percent, increasing slightly with the addition of more columns. Notably, for this dataset, the difference between ON and OFF increased as the number of entries grew. Similar to the *Eurostat* and CO2 emission datasets, renaming columns had no discernible impact on the resulting file size. Memory consumption remained constant despite an increase in the number of renamed columns within a given set of diagrams. Finally, there was a less-than-linear relationship between processed entries and memory usage. There was an approximate 1.83-fold increase in memory requirements when processing double the entries, indicating a sublinear relationship between entries and memory requirements.

4.2.2 Effect on Computational Time

This section provides a detailed analysis of the time taken to process the original datasets under different scenarios. We conduct these experiments by changing the number of entries of each dataset and calculating mean and standard deviation within the plots. The computational mean times of Eurostat, CO2 emission and private datasets are shown in figures 4.4, 4.6and 4.8, respectively. Similarly, the computational time variations in terms of standard deviation for Eurostat, CO2 emission and private datasets are shown in Figures 4.5, 4.7 and 4.9, respectively. The variables used for this analysis are the same as for the memory occupation analysis, such as varying number of entries, number of columns renamed and/or added. Running performance tests was similar to running memory occupation tests and conducted as shown below:

- First, the data was prepared for inserting into the raw database. For this, the amount of entries in the dataset were reduced to 50, 100 and 120 or 150. Each of these amounts of entries were saved in a separate CSV file.
- Second, the prepared CSV file with the right amount of entries for the required test run was inserted into the Raw Data Database. Since the process of inserting

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

data into the raw data database was not the subject under investigation, the data could be imported with the help of SQL statements.

- Third, before sending the HTTP request, which starts the harmonization of the data, a timestamp was taken
- Fourth, the data harmonization was started, for which a data harmonization HTTP request was sent to the `Data Preprocessing Microservice`. The parameters for this request were set according to the test case at hand - these are the variables mentioned before.
- Fifth, after the data harmonization had finished, another timestamp was taken and the elapsed time for the execution calculated.
- Sixth, the steps to send the HTTP request and calculate elapsed time was repeated for a hundred times for each variable combination.
- Finally, the whole process was repeated for all the other variable combinations and the results recorded.

Figures 4.4 to 4.9 shows variations on computational time for the different data sets in both mean and standard deviations, with multiple parameters as summarized below.

- On the x-axis: number of data entries. Tests with three different amount of entries were conducted - with 50, 100 and 120 or 150.
- On the y-axis: mean computational time (for the figures 4.4, 4.6 and 4.8) and Standard deviation of computational time (for the figures 4.5, 4.7 and 4.9). The given values are displayed in seconds (s).
- In each plot in Figure 4.4 to 4.9 contains two metrics (lines) such as - the blue line (ON) shows the computational time for the proposed harmonization while eliminating duplicates and empty entries. The red line (OFF) indicates the proposed harmonization while not-enabling duplicate and empty entry removal.
- We also estimate the variations, as we rename columns to the existing data base during harmonization process. We considered these variations between 1, 3 and 5. In each plot in Figure 4.4 to 4.9, the sub figures (a), (d) and (g) considered renaming a single column, the sub figures (b), (e) and (h) considered renaming three columns, whereas the sub figures (c), (f) and (i) considered renaming five columns.
- Along with the renaming, we also considered adding new columns during harmonization. We considered adding new column variations between 1, 3 and 5, similar to the renaming. In each plot in Figure 4.4 to 4.9, the sub figures (a), (b) and (c) considered adding a single column, the sub figures (d), (e) and (f) considered adding three new columns, whereas the sub figures (g), (h) and (i) considered adding five new columns.
- In summary, figure 4.4(a) is the result of adding one new column and renaming one column. Similarly, figure 4.4(b) shows adding one new column while renaming three columns. Figure 4.4(c) shows adding one new column while renaming five existing columns and so on.

The results for mean values and their corresponding standard deviations of varying com-

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

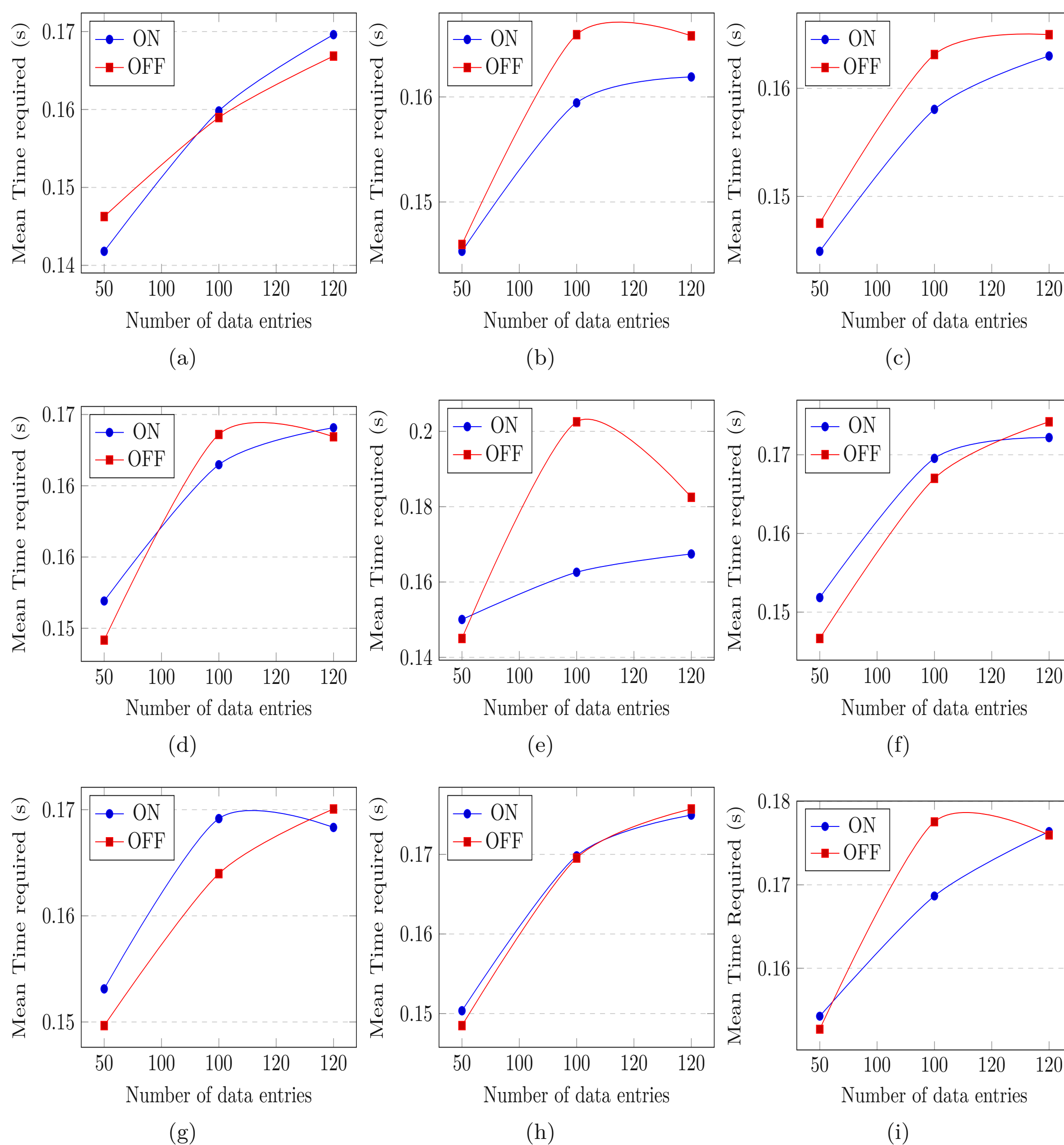


Figure 4.4: Time required mean values - Eurostat data set

putational time in *Eurostat* data set were shown in figure 4.4 and figure 4.5, respectively. In figure 4.4 and figure 4.5, there are two plots displayed - ON and OFF. These plots, the same as before, correspond to the empty and double entry removal turned on (ON) or off (OFF).

Figure 4.4 shows that, depending on the amount of data entries, the mean execution

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek Your knowledge hub WIEN

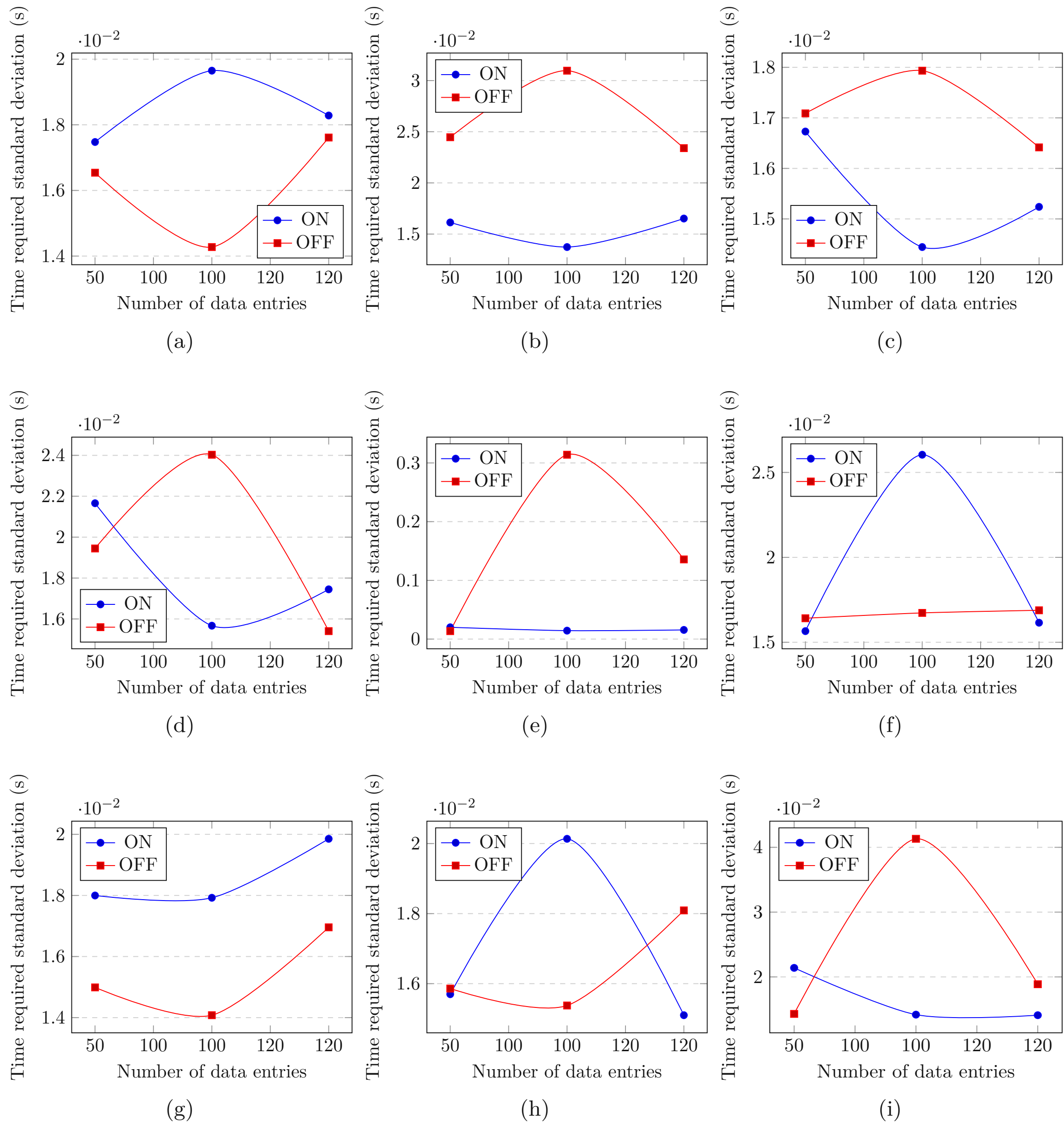


Figure 4.5: Time required standard deviation - Eurostat data set

time in almost all cases increases. The execution time increases by a factor of around ten percent by increasing the amount of entries from 50 to 100. Furthermore, by analyzing the execution curves in detail and comparing them, it can be seen that executions with double and empty entry removal turned on (ON) will take less time to run than the same runs with double and empty entry removal turned off (OFF). The difference here lies at around 1.12 %, thus on average the ON execution requires 1.12 % less time than its

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

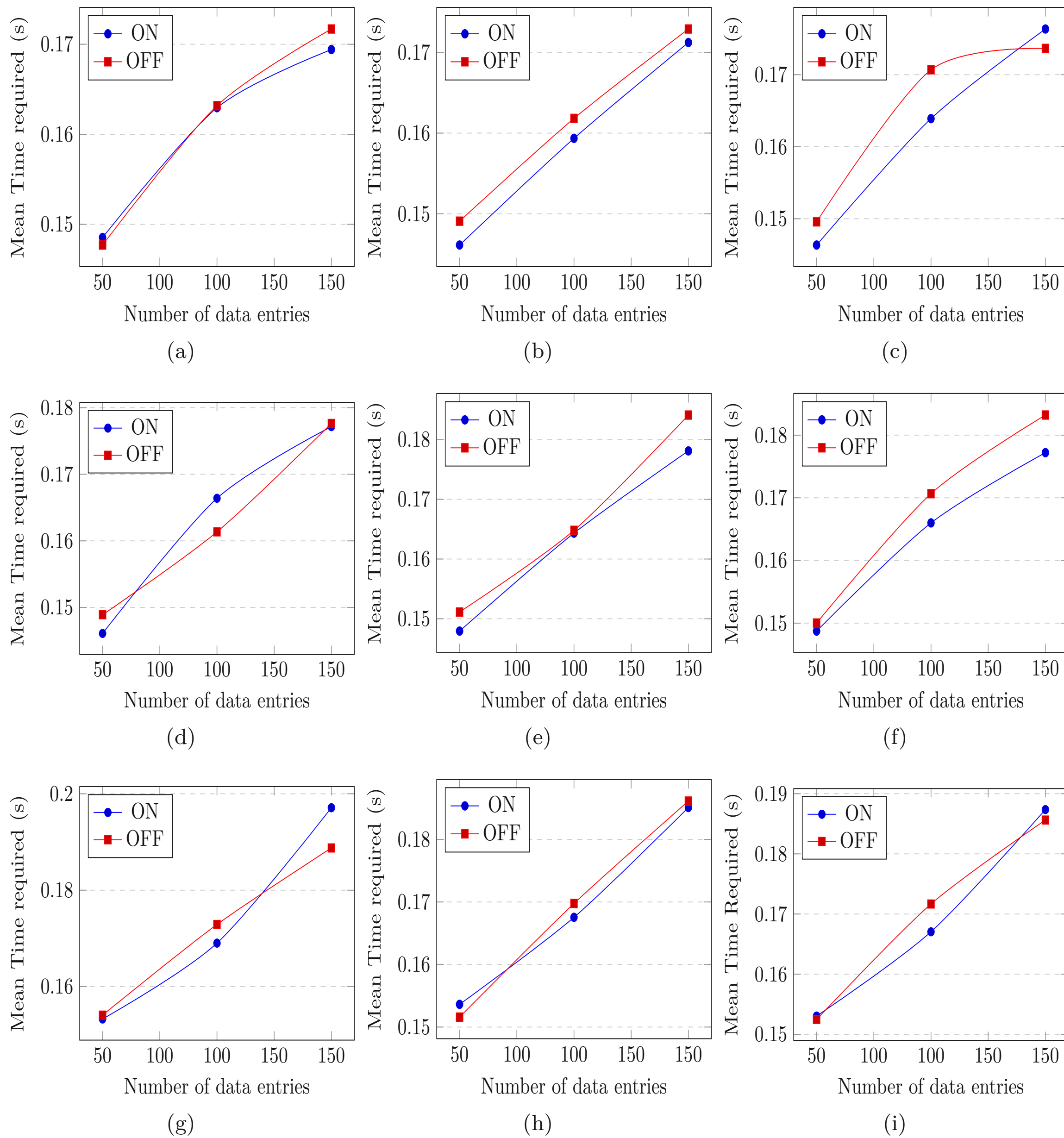


Figure 4.6: Time required mean values - CO2 data set

counterpart OFF execution. When analyzing the performance differences for adding new columns, adding new columns increased the runtime by around two to three percent. For renaming columns, the increase in runtime was conducted only around 0,1 %. Similarly, standard deviation plots can be viewed in figure 4.5 and it shows that for the runs differing standard deviations can be observed, mostly in the range of 0,01 to 0,04 seconds.

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek Your knowledge hub WIEN

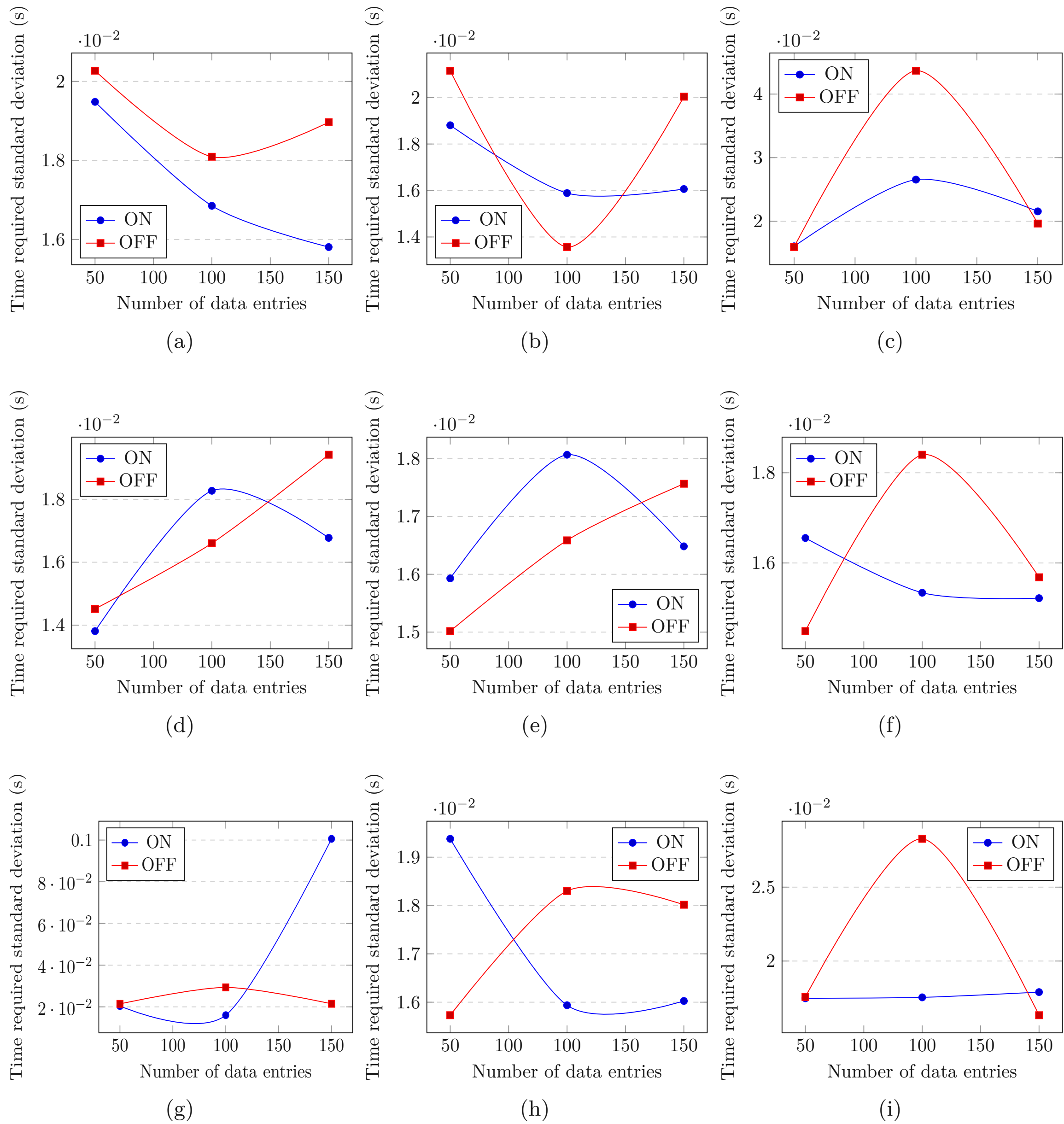


Figure 4.7: Time required standard deviation - CO2 data set

Figure 4.6 shows the mean values for the *CO2 emission* data set. First and foremost it can be seen that, by enhancing the amount of data entries, the mean execution time improved in almost all cases. The execution time increases by around ten percent by increasing the amount of entries from 50 to 100. Furthermore, by analyzing the execution curves in detail and comparing them, it can be seen that executions with double and empty entry removal turned on (ON) will take less time to run than the same runs with double and

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Wien Bibliothek Your knowledge hub

empty entry removal turned off (OFF). The difference here lies at around 0.8 %, thus on average the ON execution requires 0.8 % less time than its counterpart OFF execution. When analyzing the performance differences for adding new columns, adding new columns increased the runtime by around two to three percent. For renaming columns, there was no significant difference in runtime. Similarly, standard deviation plots can be viewed in figure 4.7 and it shows that for the runs differing standard deviations can be observed, mostly being in the range of 0,01 to 0,04 seconds, similarly to the *Eurostats* datasets standard deviation.

Figure 4.8 shows the mean computational time variations for the private data set. The mean execution time increases in all cases when the number of data entries is increased. The execution time is extended by a factor of around 20 % for expanding the amount of entries from 50 to 100. Furthermore, by analyzing the execution curves in detail and comparing them, it can be seen that executions with double and empty entry removal turned on (ON) spend less time running than the same runs with double and empty entry removal turned off (OFF). The difference here lies around 1.1 %, thus on average the ON execution consumes 1.1 % less time than its counterpart OFF execution. When analyzing the performance differences for adding new columns, adding new columns prolonged the runtime by around two to three percent. For renaming columns, the time required for changing more columns took less time than renaming only a few columns. The increase ranged from 0.6 % to 1.1 %. Similarly, the standard deviation for computational time variations in private datasets is shown in figure 4.9. Here it can be seen that for the runs differing standard deviations can be observed. For this data set the standard deviations were between 0.02 and 0.1 seconds, indicating that they were significantly higher than for the previously explained data sets. Furthermore, for most plots, the standard deviation increases rapidly, as the number of entries increases.

In addition to running performance tests for datasets, where the raw data is situated in the Raw Data database, additional performance tests were run with the raw data files placed in blob storage. Thus, instead of requesting raw data from a database, files with different amounts of data were used. The variables and the steps remained the same as explained in the previously described time-required performance tests. The data sets used were private data files. The figure 4.10 displays the mean values for the private data set. It is clearly evident that, when the number of data entries is increased, the mean execution time increases. The execution time increases by a factor of around 12 to 15 % for raising the amount of entries from 50 to 100. Furthermore, by analyzing the execution curves in detail and comparing them, it can be seen that executions with double and empty entry removal turned on (ON) required more time to run than the same runs with double and empty entry removal turned off (OFF). In contrast to the previous results, where ON data runs were faster, the OFF data runs were on average about 0.3 percent faster than their ON counterparts. When analyzing the performance differences for adding new columns, adding new columns increased the runtime by around 0.6 %. For renaming columns, the time required for renaming more columns took in average 1 to almost 3 % more than renaming less columns.

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

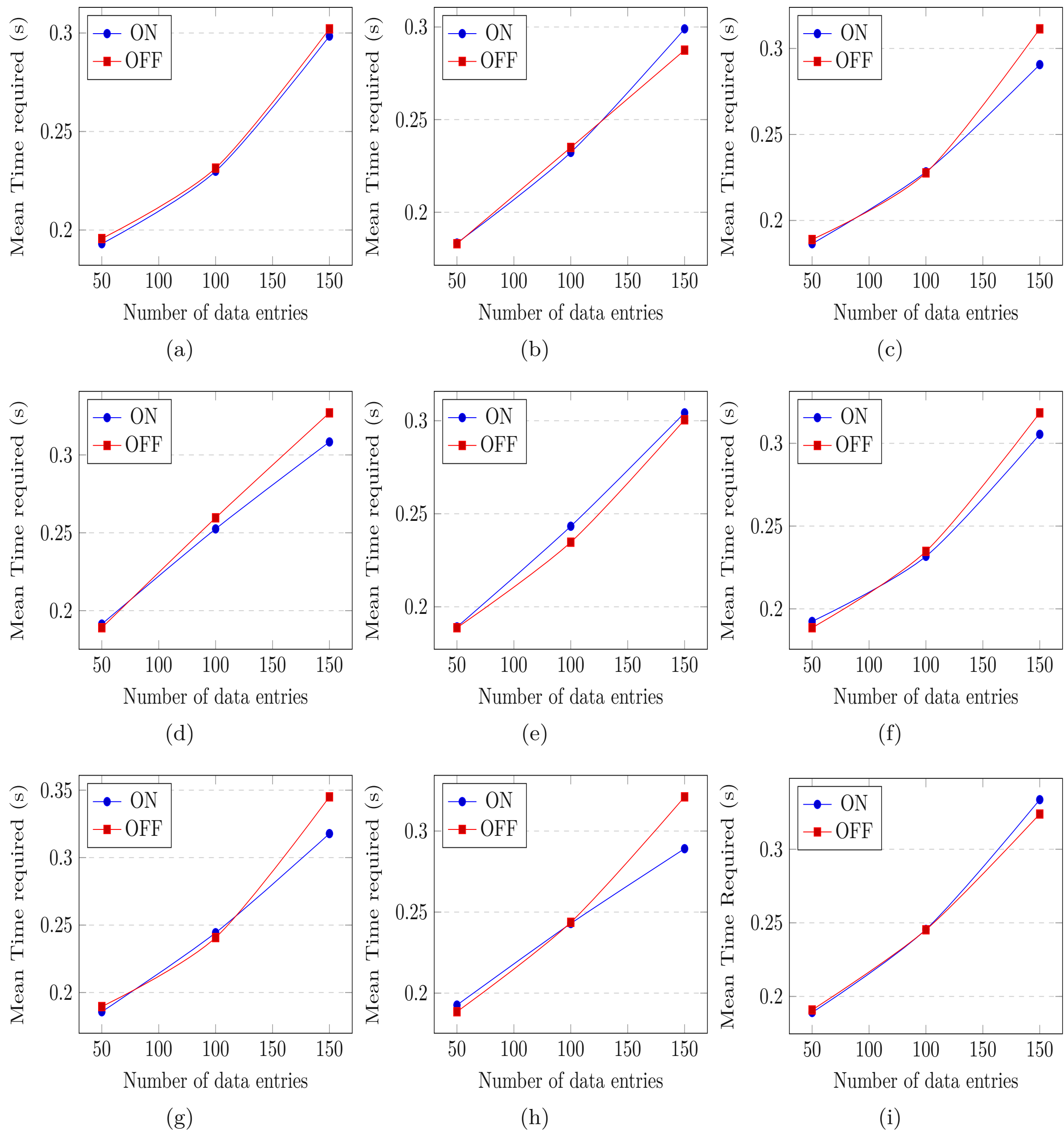


Figure 4.8: Time required mean values - private data set

Finally, standard deviation plots can be viewed in the following figure 4.11. As can be seen here, there are differences between the standard deviations for the runs. The standard deviations for this data set ranged from 0.06 to 0.12 seconds, a significantly higher standard deviation than those for the previously explained data sets. It is important to note that the highest standard deviation was observed for fewer entries. This opposes the previously observed results, that with an increasing amount of entries, the standard

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek WIEN Your knowledge hub

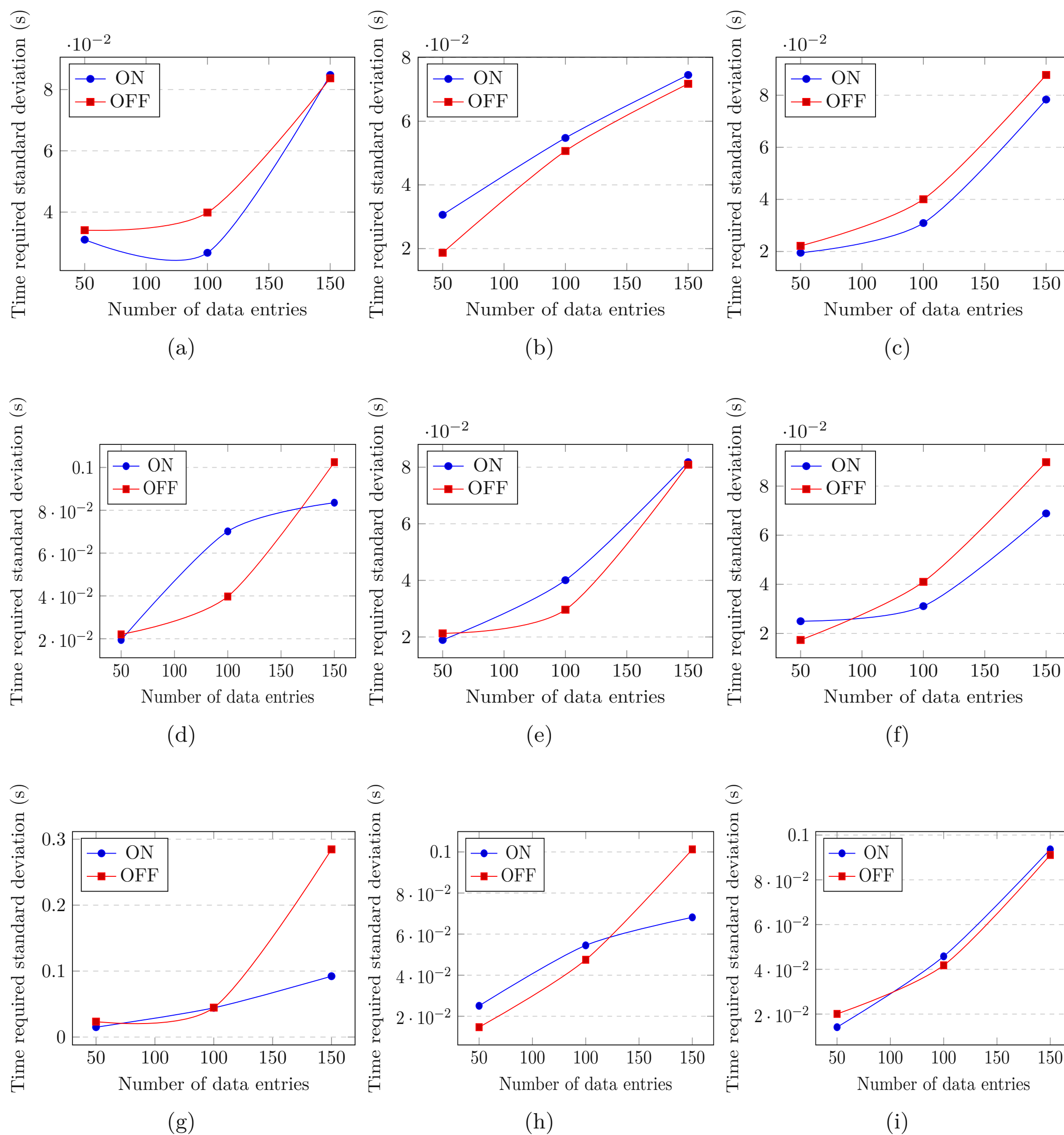


Figure 4.9: Time required standard deviation - private data set

deviation also increases.

4.2.3 Data Analysis along with Harmonization

In preparation for the analysis, both publicly available and private data were imported into the system. The publicly available data was sourced from *Eurostat* and Trading

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

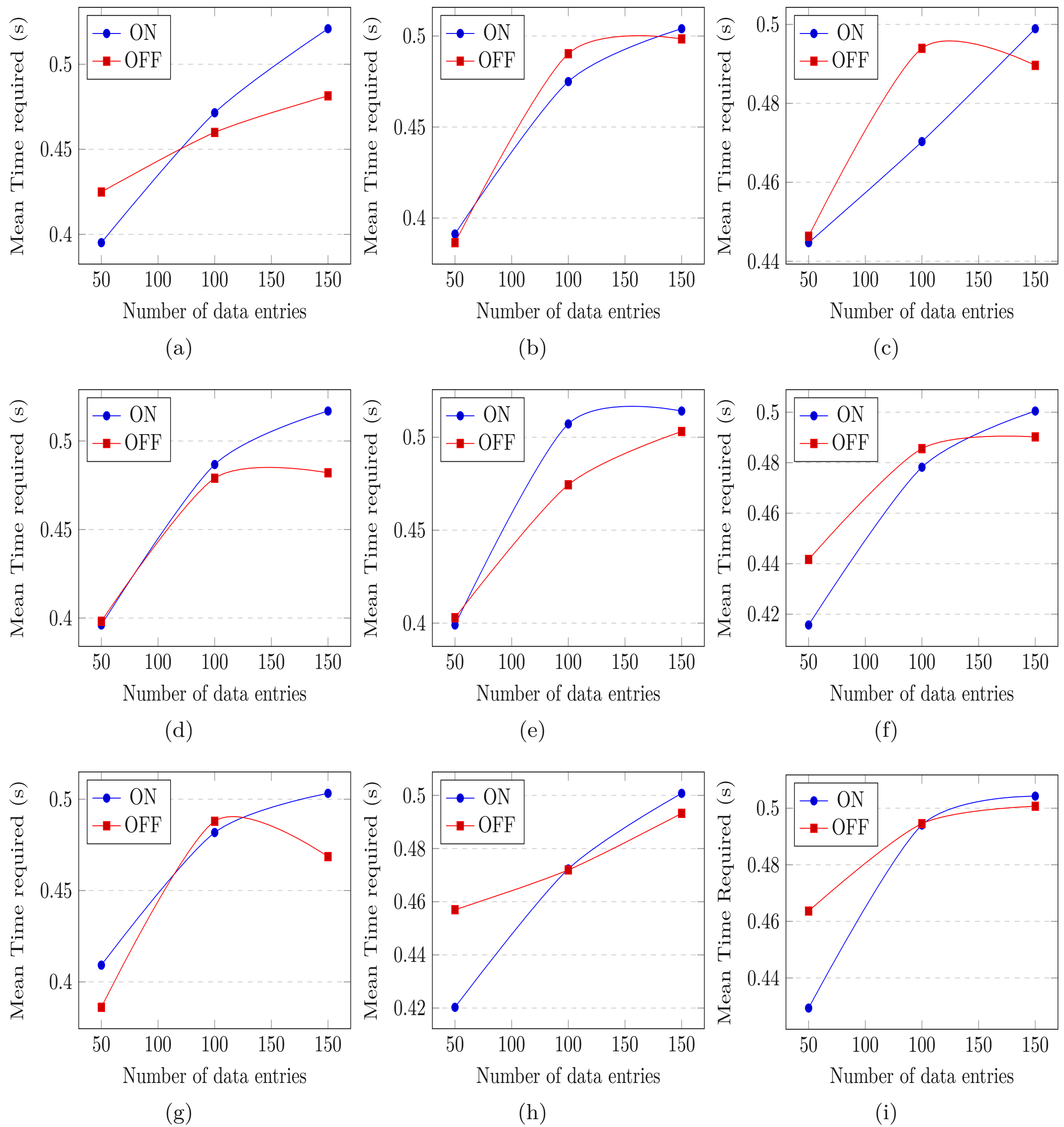


Figure 4.10: Time required mean values - private data set blob storage

Economics, which are summarized below.

- *Eurostat* data where national accounts aggregates by industry are displayed [55].
- *Trading Economics* [56] datasets were extracted below:
  - CO2 Emissions [57]
  - Precipitation [58]

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

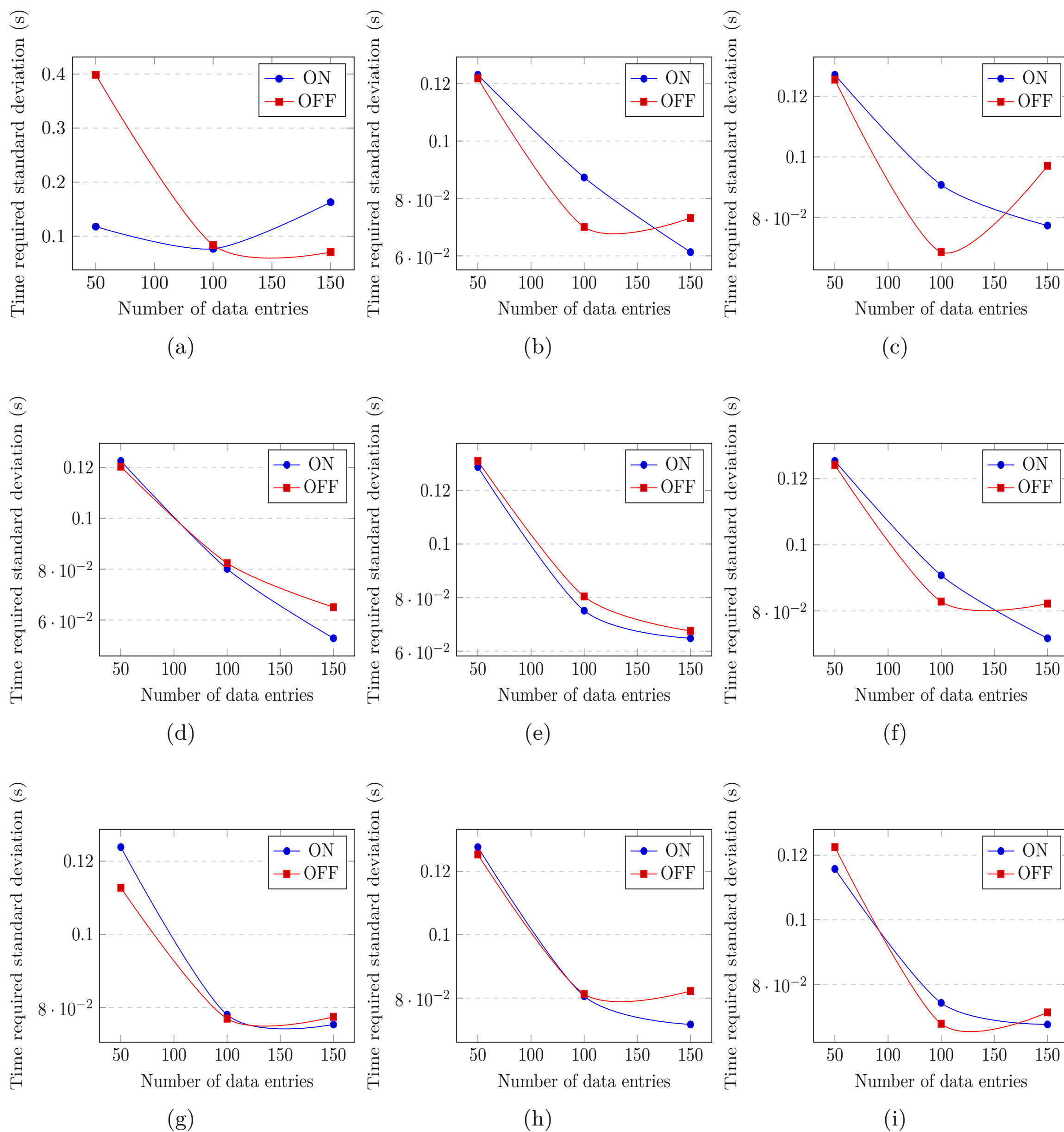


Figure 4.11: Time required standard deviation - private data set blob storage

– Temperature [59]

In addition to public data, we also acquired industry data to verify the effectiveness of the proposed harmonization system. This dataset includes data from 2022 with the different suppliers, product groups, delivered products and services together with the prices for these efforts. Due to an agreement with the industry partner, we cannot provide more

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

details about these datasets explicitly.

After this data was imported into the system and harmonized, the process of generating new knowledge with PowerBI data analysis from this data could be started. A summary of these findings can be found in table 4.6. The private customer data product group was linked to different industries in Eurostat's data. As can be seen, both product groups "22100000" and "25139998" belong to the "Construction" data industry, while the product group "25160000" can be linked to "Financial and insurance activities" etc.

Table 4.6: Link between private data product group and Eurostat data industry

Customer product group nr	**Eurostat data industry**
22100000	Construction
25139998	Construction
25160000	Financial and insurance activities
26040100	Electricity, gas, steam and air conditioning supply
33049091	Electricity, gas, steam and air conditioning supply
33100000	Electricity, gas, steam and air conditioning supply
37070000	Manufacture of other non-metallic mineral products

The industry partner provided data on CO2 emissions in tons per Euro for industries not covered by Eurostat. These emissions were correlated with Eurostat's industries, enabling total emissions per industry annually. The corresponding calculations are detailed in Table 4.8. The CO2 emissions of various products were calculated based on the results of the aforementioned tables (see Table 4.7). The first column denotes unique product group identifiers, while the second column presents the net value of each purchased product or service. Subsequently, the estimated tonnes of CO2 emissions for different industries from Table 4.8 were incorporated. The final column illustrates the total CO2 emissions attributed to each product or service. For example, the initial product with the identifier 33100000 has a cost per order value of 2,070,814.36 EUR. Since the estimated CO2 emissions per Euro for the industry "Electricity, gas, steam and air conditioning supply" is 0.001006, the resulting CO2 emissions estimate amounts to nearly 2072 tonnes for this particular order. The successful implementation of the architecture, facilitating the consolidation of diverse data sources, data importation, harmonization and subsequent analysis, validates the versatility of this approach for large-scale data analysis.

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

Table 4.7: Results CO2 emissions per product

Product group	Order net value	CO2 emissions/€	CO2 emissions calc.
33100000	2 070 814,36	0,0010005664	2071,987327
33100000	2 058 920,44	0,0010005664	2060,08667
25139998	2 000 000,00	0,0000431778	86,35562291
37070000	1 962 612,26	0,0020716748	4065,894437
33049091	1 960 997,00	0,0010005664	1962,107763
25160000	1 944 823,91	0,0000034574	6,724017774
25160000	1 836 882,96	0,0000034574	6,350823644
22100000	1822382	0,0000431778	78,68646639
33049091	1787280	0,0010005664	1788,292365
25139998	1775050	0,0000431778	76,64277422
26040100	1696543,72	0,0010005664	1697,50469
22100000	1600000	0,0000431778	69,08449833
22100000	1600000	0,0000431778	69,08449833
33049091	1562394,44	0,0010005664	1563,279424
33100000	1550500	0,0010005664	1551,378246
	27229201,09		**17153,45962**

Table 4.8: Emissions per Euro per Industry

Industry	Volume in industry (€)	CO2 Emissions/Yr	CO2 emissions/€
Electricity, gas, steam and air conditioning supply	6 098 200 000	6 101 654,19	0,001000566
Construction	26 329 800 000	1 136 863,14	0,000043178
Manufacture of other non-metallic mineral products	2 679 500 000	5 551 052,73	0,002071675
Financial and insurance activities	15 589 400 000	53 898,66	0,000003457

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

CHAPTER 5

Conclusion

This thesis presents, implements, and evaluates a microservices-based data harmonization architecture. The proposed architecture supports multiple aspects of data analysis - data gathering, data ingestion, data preprocessing and enabling access to data inside the system. This thesis aimed to automate the data harmonization part of this process and evaluate some possible preprocessing strategies. As the proposed architecture was implemented, performance assessment and memory analyses were conducted using three different data sets - such as two publicly available (i.e., Eurostats, and CO2 emissions), one closed and private data set. It is generally true that importing data from a database is significantly faster than importing data from a file. Furthermore, it can be seen that removing empty and double data entries reduces memory consumption and in most cases also the processing time. Therefore, removing empty and double entries during data preprocessing will have a significant impact on resource consumption. It is proved through our experiments that the proposed architecture automates data preprocessing in an extremely reliable and fast way by utilizing this and similar systems. Our work can be extended to advance preprocessing techniques, enhance scalability, integrate with advanced analytics, address privacy concerns, and optimize microservice-based data harmonization architectures in real-world deployments in the future.

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.

List of Figures

3.1 Overview of System Architecture for Harmonization 12
3.2 Public Raw Data Import Process Diagram 14
3.3 Raw Data Extractor Class Diagram 15
3.4 Raw Data Request Parameter Microservice Class Diagram 18
3.5 Public Raw Data Microservice Class Diagram 21
3.6 Public Raw Data Preprocessing working Process 22
3.7 Private Raw Data Preprocessing 23
3.8 Preprocessed Public Data Microservice Class Diagram 24
3.9 Preprocessed Private Data Microservice Class Diagram 25
3.10 Proposed Data Access Process 26
3.11 Sequence Diagram for data extraction 27
3.12 Preprocess Raw Public Data Sequence Diagram 30
3.13 Preprocess Raw Private Data Sequence Diagram 31
3.14 Data Preprocessing Microservice Class Diagram 36

4.1 Memory occupied for Eurostat data set 55
4.2 Memory occupied - CO2 emissions 57
4.3 Memory occupied - private data dataset 58
4.4 Time required mean values - Eurostat data set 61
4.5 Time required standard deviation - Eurostat data set 62
4.6 Time required mean values - CO2 data set 63
4.7 Time required standard deviation - CO2 data set 64
4.8 Time required mean values - private data set 66
4.9 Time required standard deviation - private data set 67
4.10 Time required mean values - private data set blob storage 68
4.11 Time required standard deviation - private data set blob storage 69

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.

List of Tables

4.1 Trading Economics data model 51
4.2 Eurostat data model 51
4.3 Trading Economics preprocessed data model 52
4.4 Eurostat preprocessed data model 52
4.5 Industry partner private data model 53
4.6 Link between private data product group and Eurostat data industry . . 70
4.7 Results CO2 emissions per product 71
4.8 Emissions per Euro per Industry 71

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.

Bibliography

[1] T. A. Runkler, *Data analytics.* Springer, 2020.

[2] C.-W. Tsai, C.-F. Lai, H.-C. Chao, and A. V. Vasilakos, *Big data analytics.* 2016.

[3] ——, „Big data analytics: A survey“, *Journal of Big data*, vol. 2, pp. 1–32, 2015.

[4] Y. Chen, H. Chen, A. Gorkhali, Y. Lu, Y. Ma, and L. Li, „Big data analytics and big data science: A survey“, *Journal of Management Analytics*, vol. 3, no. 1, pp. 1–42, 2016.

[5] P. K. Donta, B. Sedlak, V. Casamayor Pujol, and S. Dustdar, „Governance and sustainability of distributed continuum systems: A big data approach“, *Journal of Big Data*, vol. 10, no. 1, p. 53, 2023.

[6] M. Blackwell. „Data is the new oil... so to speak“. (), [Online]. Available: `https://www.corelogic.uk/news/data-is-the-new-oil-so-to-speak/`.

[7] Z. Rasheed, A. Raza, R. Gholami, M. Rabiei, A. Ismail, and V. Rasouli, „A numerical study to assess the effect of heterogeneity on co2 storage potential of saline aquifers“, *Energy Geoscience*, vol. 1, no. 1-2, pp. 20–27, 2020.

[8] P. Friedlingstein, M. O’sullivan, M. W. Jones, *et al.*, „Global carbon budget 2022“, *Earth System Science Data Discussions*, vol. 2022, pp. 1–159, 2022.

[9] „Internet of things and smart cities“. (), [Online]. Available: `https://dsg.tuwien.ac.at/research.html`.

[10] L. Chen, J. J. Dolado, J. Gonzalo, and A. Ramos, „Heterogeneous predictive association of co2 with global warming“, *Economica*, vol. 90, no. 360, pp. 1397–1421, 2023.

[11] Z. Zhao, T. Yuan, X. Shi, and L. Zhao, „Heterogeneity in the relationship between carbon emission performance and urbanization: Evidence from china“, *Mitigation and Adaptation Strategies for Global Change*, vol. 25, pp. 1363–1380, 2020.

[12] G. M. Karam and R. J. A. Buhr, „Starvation and critical race analyzers for ada“, *IEEE Transactions on Software Engineering*, vol. 16, no. 8, pp. 829–843, 1990.

[13] S. Singhal and M. Jena, „A study on weka tool for data preprocessing, classification and clustering“, *International Journal of Innovative technology and exploring engineering (IJItee)*, vol. 2, no. 6, pp. 250–253, 2013.

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

[14] „Big data what it is and why it matters“. (), [Online]. Available: https://www.sas.com/en_us/insights/big-data/what-is-big-data.html.

[15] G. Kumar, S. Basri, A. A. Imam, S. A. Khowaja, L. F. Capretz, and A. O. Balogun, „Data harmonization for heterogeneous datasets: A systematic literature review“, *Applied Sciences*, vol. 11, no. 17, p. 8275, 2021.

[16] D. Doiron, P. Burton, Y. Marcon, *et al.*, „Data harmonization and federated analysis of population-based studies: The BioSHaRE project“, *Emerging themes in epidemiology*, vol. 10, pp. 1–8, 2013.

[17] P. Granda and E. Blasczyk, „Data harmonization“, *Guidelines for Best Practice in Cross-Cultural Surveys, 2nd edn. Ann Arbor, MI: Survey Research Center, Institute for Social Research, University of Michigan*, 2010.

[18] C. Cheng, L. Messerschmidt, I. Bravo, *et al.*, „A general primer for data harmonization“, *Scientific Data*, vol. 11, no. 1, p. 152, 2024.

[19] M. Mazzara, N. Dragoni, A. Bucchiarone, A. Giaretta, S. T. Larsen, and S. Dustdar, „Microservices: Migration of a mission critical system“, *IEEE Transactions on Services Computing*, vol. 14, no. 5, pp. 1464–1477, 2018.

[20] H. Zhao, S. Deng, Z. Liu, J. Yin, and S. Dustdar, „Distributed redundant placement for microservice-based applications at the edge“, *IEEE Transactions on Services Computing*, vol. 15, no. 3, pp. 1732–1745, 2020.

[21] A. Bucchiarone, N. Dragoni, S. Dustdar, *et al.*, „Microservices“, *Science and Engineering. Springer*, 2020.

[22] S. Newman, *Building microservices.* " O'Reilly Media, Inc.", 2021.

[23] J. Thönes, „Microservices“, *IEEE software*, vol. 32, no. 1, pp. 116–116, 2015.

[24] „What is a distributed system?“ (), [Online]. Available: https://www.atlassian.com/microservices/microservices-architecture/distributed-architecture.

[25] I. Corporation. „Federated systems“. (), [Online]. Available: https://www.ibm.com/docs/en/db2oc?topic=SSFMBX/com.ibm.data.fluidquery.doc/topics/cfpint01.htm.

[26] J. Ambite and C. Knoblock, „Agents for information gathering“, *IEEE Expert*, vol. 12, no. 5, pp. 2–4, 1997. DOI: 10.1109/64.621219.

[27] J. Lloret, J. Tomas, A. Canovas, and L. Parra, „An integrated iot architecture for smart metering“, *IEEE Communications Magazine*, vol. 54, no. 12, pp. 50–57, 2016. DOI: 10.1109/MCOM.2016.1600647CM.

[28] M. Wu, T.-J. Lu, F.-Y. Ling, J. Sun, and H.-Y. Du, „Research on the architecture of internet of things“, in *2010 3rd International Conference on Advanced Computer Theory and Engineering(ICACTE)*, vol. 5, 2010, pp. V5-484-V5–487. DOI: 10.1109/ICACTE.2010.5579493.

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

[29] S. Pudlewski and K. Landers, „Dima: Distributed iot modeling architecture", in *2021 IEEE International Conference on Communications Workshops (ICC Workshops)*, 2021, pp. 1–6. DOI: 10.1109/ICCWorkshops50388.2021.9473662.

[30] M. Richards. „Software architecture patterns". (), [Online]. Available: https://isip.piconepress.com/courses/temple/ece_1111/resources/articles/20211201_software_architecture_patterns.pdf.

[31] R. Chen, S. Li, and Z. Li, „From monolith to microservices: A dataflow-driven approach", in *2017 24th Asia-Pacific Software Engineering Conference (APSEC)*, 2017, pp. 466–475. DOI: 10.1109/APSEC.2017.53.

[32] C. Richardson. „What are microservices?" (), [Online]. Available: https://microservices.io/.

[33] R. P. d. C. C. Macedo. „Implementation of a data lake in a microservices architecture". (), [Online]. Available: http://hdl.handle.net/10451/63925.

[34] M. Parciak, M. Suhr, C. Schmidt, *et al.*, „Fairness through automation: Development of an automated medical data integration infrastructure for fair health data in a maximum care university hospital", *BMC Medical Informatics and Decision Making*, vol. 23, no. 1, p. 94, May 2023, ISSN: 1472-6947. DOI: 10.1186/s12911-023-02195-3. [Online]. Available: https://doi.org/10.1186/s12911-023-02195-3.

[35] G. Kumar, S. Basri, A. A. Imam, S. A. Khowaja, L. F. Capretz, and A. O. Balogun, „Data harmonization for heterogeneous datasets: A systematic literature review", *Applied Sciences*, vol. 11, no. 17, 2021, ISSN: 2076-3417. DOI: 10.3390/app11178275. [Online]. Available: https://www.mdpi.com/2076-3417/11/17/8275.

[36] X. Zhao, S. Coxe, M. H. Sibley, C. Zulauf-McCurdy, and J. W. Pettit, „Harmonizing depression measures across studies: A tutorial for data harmonization", *Prevention Science*, vol. 24, no. 8, pp. 1569–1580, Nov. 2023, ISSN: 1573-6695. DOI: 10.1007/s11121-022-01381-5. [Online]. Available: https://doi.org/10.1007/s11121-022-01381-5.

[37] C. Marzi, M. Giannelli, A. Barucci, C. Tessa, M. Mascalchi, and S. Diciotti, „Efficacy of mri data harmonization in the age of machine learning: A multicenter study across 36 datasets", *Scientific Data*, vol. 11, no. 1, p. 115, Jan. 2024, ISSN: 2052-4463. DOI: 10.1038/s41597-023-02421-7. [Online]. Available: https://doi.org/10.1038/s41597-023-02421-7.

[38] I. W. Sampaio, E. Tassi, M. Bellani, *et al.*, „Comparison of multi-site neuroimaging data harmonization techniques for machine learning applications", in *IEEE EUROCON 2023 - 20th International Conference on Smart Technologies*, 2023, pp. 307–312. DOI: 10.1109/EUROCON56442.2023.10198911.

[39] D. R. Himadri Sikhar Khargharia Sid Shakya. „Comparative analysis of metaheuristics techniques for trade data harmonization". (), [Online]. Available: https://www.scitepress.org/Papers/2023/121766/121766.pdf.

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

[40] K. Adhikari, S. B. Patten, A. B. Patel, *et al.*, „Data harmonization and data pooling from cohort studies: A practical approach for data management", *International journal of population data science*, vol. 6, no. 1, 2021.

[41] R. D. Kush, D. Warzel, M. A. Kush, *et al.*, „FAIR data sharing: The roles of common data elements and harmonization", *Journal of biomedical informatics*, vol. 107, p. 103 421, 2020.

[42] M. E. Torbati, D. S. Minhas, G. Ahmad, *et al.*, „A multi-scanner neuroimaging data harmonization using RAVEL and ComBat", *Neuroimage*, vol. 245, p. 118 703, 2021.

[43] J. Kalter, M. G. Sweegers, I. M. Verdonck-de Leeuw, J. Brug, and L. M. Buffart, „Development and use of a flexible data harmonization platform to facilitate the harmonization of individual patient data for meta-analyses", *BMC research notes*, vol. 12, pp. 1–6, 2019.

[44] D. Firnkorn, M. Ganzinger, T. Muley, M. Thomas, and P. Knaup, „A generic data harmonization process for cross-linked research and network interaction", *Methods of information in medicine*, vol. 54, no. 05, pp. 455–460, 2015.

[45] P. Papadimitroulas, L. Brocki, N. C. Chung, *et al.*, „Artificial intelligence: Deep learning in oncological radiomics and challenges of interpretability and data harmonization", *Physica Medica*, vol. 83, pp. 108–121, 2021.

[46] A. R. Fay, L. Gregor, P. Landschutzer, *et al.*, „SeaFlux: Harmonization of air-sea CO2 fluxes from surface pCO (2) data products using a standardized approach", *Earth System Science Data*, vol. 13, no. 10, pp. 4693–4710, 2021.

[47] S. Tarazona, L. Balzano-Nogueira, D. Gómez-Cabrero, *et al.*, „Harmonization of quality metrics and power calculation in multi-omic studies", *Nature communications*, vol. 11, no. 1, p. 3092, 2020.

[48] C. Richardson. „Communication patterns". (), [Online]. Available: `https://microservices.io/patterns/index.html#communication-patterns`.

[49] ——, „Pattern: Messaging". (), [Online]. Available: `https://microservices.io/patterns/communication-style/messaging.html`.

[50] ——, „Pattern: Remote procedure invocation (rpi)". (), [Online]. Available: `https://microservices.io/patterns/communication-style/rpi.html`.

[51] C. Cheng, L. Messerschmidt, I. Bravo, *et al.*, „A general primer for data harmonization", *Scientific Data*, vol. 11, no. 1, p. 152, 2024.

[52] Microsoft. „Authenticate with azure container registry from azure kubernetes service". (), [Online]. Available: `https://learn.microsoft.com/en-us/azure/aks/cluster-container-registry-integration?tabs=azure-cli`.

[53] kubernetes.io. „Service". (), [Online]. Available: `https://kubernetes.io/docs/concepts/services-networking/service/`.

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

[54] ——, „Ingress“. (), [Online]. Available: https://kubernetes.io/docs/concepts/services-networking/ingress/.

[55] t. s. o. o. t. E. U. Eurostat. „National accounts aggregates by industry (up to nace a*64)“. (), [Online]. Available: https://ec.europa.eu/eurostat/databrowser/view/NAMA_10_A64__custom_6414125/default/table?lang=en.

[56] T. Economics. „Indicators“. (), [Online]. Available: https://tradingeconomics.com/indicators.

[57] ——, „Co2 emissions“. (), [Online]. Available: https://tradingeconomics.com/country-list/co2-emissions.

[58] ——, „Average precipitation by country“. (), [Online]. Available: https://tradingeconomics.com/country-list/precipitation.

[59] ——, „Average temperature by country“. (), [Online]. Available: https://tradingeconomics.com/country-list/temperature.

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub